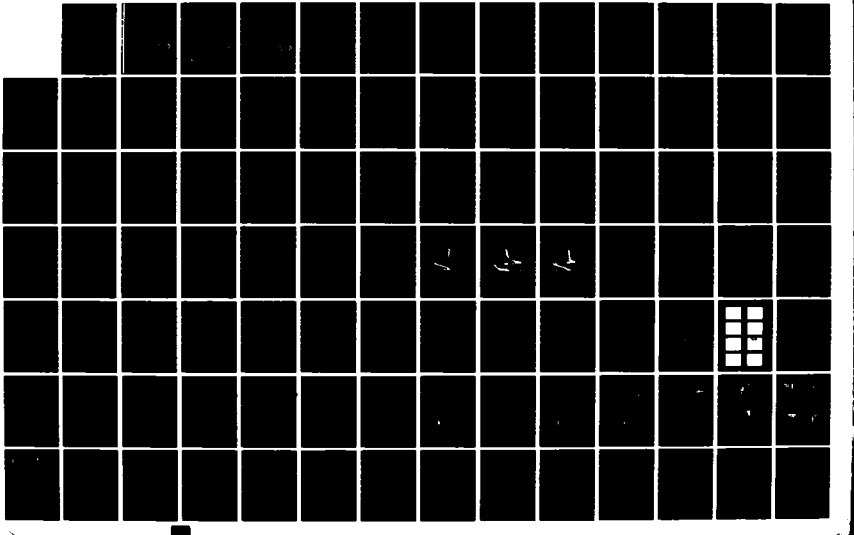
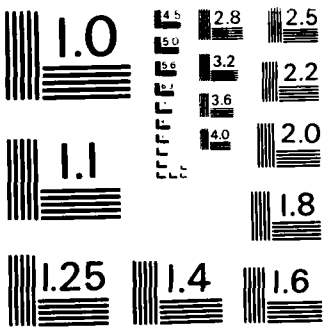
AD-A139 628 PLASMA PHYSICS ISSUES IN ADVANCED SIMULTTION RESEARCH 1/2
(U) SCIENCE APPLICATIONS INC MCLEAN VA A MONDELLI
01 NOV 83 SAI-84-235-WA SBI-AD-E001 641
UNCLASSIFIED N00014-81-C-2041 F/G 20/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

PLASMA PHYSICS ISSUES IN ADVANCED SIMULATION RESEARCH

FINAL REPORT NO. SAI-84-235-WA

Alfred Mondelli

SCIENCE APPLICATIONS, INC.

PLASMA PHYSICS ISSUES IN ADVANCED SIMULATION RESEARCH

FINAL REPORT NO. SAI-84-235-WA

Alfred Mondelli

SCIENCE APPLICATIONS, INC.

Post Office Box 1303, 1710 Goodridge Drive, McLean, Virginia 22102, (703) 821-4300

PLASMA PHYSICS ISSUES IN ADVANCED SIMULATION RESEARCH

SAI-84-235-WA

FINAL REPORT

Submitted to

Plasma Physics Division
Naval Research Laboratory
Washington, D.C. 20375

Prepared Under:

Contract No. N00014-81-C-2041

Prepared by:

A. Mondelli
Science Applications, Inc.
McLean, VA 22102

NOVEMBER 1, 1985

SCIENCE APPLICATIONS, INCORPORATED
1710 Goodridge Drive, McLean, Virginia 22102 (703)734-5840

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

This document is the final report for Contract Number N00014-81-C-2041, which covered work performed for Code 4707 of the Naval Research Laboratory (NRL) by the Plasma Physics Division of Science Applications, Inc. (SAI) during the period 24 November 1980 to 24 September 1982.

The material covered in this report consists of three general areas in which plasma physics plays a significant role in the modeling of radiation sources for the advanced simulation research program sponsored by the Defense Nuclear Agency (DNA). The first is the description of a basic model for the implosion of a system of identical wires driven by a pulsed-power generator. The second is a model for computing the linear ideal MHD instability growth rates for azimuthally-symmetric, cylindrical Z-pinch equilibria. This analysis includes both kink and sausage-type perturbations of the equilibrium. The third area concerns the properties of magnetically-insulated power feeds for driving imploding Z-pinch loads.

SECTION 2

THE "WIRES" CODE -- A SIMPLE MODEL FOR IMPLOSIONS

It has been known for several years that imploding wire arrays during run-in obey the so-called "F-ma" dynamics to remarkable accuracy.[1] Recent data has suggested that for truly massive arrays ($\gtrsim$ 500 μg) the F = ma scaling finally breaks down, with arrays apparently going unstable prior to achieving a significant inward acceleration. For arrays of < 300 μg, however, the F = ma formalism appears good, and predicts the assembly time to within a few nanoseconds.[2] Given that the dynamics of the wire centers has been understood at this level, it appears reasonable to attempt a generalization of the F = ma model to include a radiation algorithm and a prescription for tracking the internal energy of the wires.

One strong motivation for pursuing this type of model for the early-time behavior of wire arrays is to establish the correct initial conditions for one-dimensional, radiation-coupled hydro codes, such as WHYRAD and SPLATT, which currently assume that the individual wires instantly expand into a plasma annulus with a temperature of 1-10 eV prior to implosion. By calculating the intial conditions from the generalized F = ma model described here, these codes will be able to provide scaling with

number of wires varied and total array mass fixed, which currently is not possible to compute. Also, the results of many runs of WHYRAD and SPLATT indicate that during this early implosion period, the radiation is basically a black-body spectrum, and the plasma remains relatively cool, indicating that the detailed radiation and chemistry package of these sophisticated codes is not needed for the run-in phase of the implosion. By using the simple model described here, the early-time behavior of the array can be obtained in a small fraction of the computer time required in WHYRAD, thereby allowing the detailed models to be utilized where they are most necessary, during the assembly and compression of the plasma annulus on axis.

Figure 2-1 shows a schematic representation of the wire array at time t. The individual wires have radius, $a(t)$, the wire array has radius, $r(t)$, and the entire system of N wires is enclosed by a cylinder of radius, b, which carries the return current. The external circuit is shown in Figure 2-2, and consists of an external voltage generator, providing a voltage waveform, $V(t)$, with a generator impedance Z_o. This generator section could be replaced by a transmission line of impedance, Z_o and length, τ, which is initially fully charged. The generator drives a time-dependent load described by the diode-housing inductance, L_D, and the time-varying plasma resistance and inductance, $R_p(t)$ and $L_p(t)$ respectively.

The plasma circuit parameters are assumed to be correctly given by the Russel formula for inductance and the Spitzer resistivity,

$$L_p = \frac{\ell}{2N} + \frac{2\ell}{N} \ell n \left(\frac{b^N}{Na r^{N-1}}\right) \quad nH$$

for lengths in centimeters, where ℓ is the array length, and

$$R_p = \frac{\eta \ell}{N \pi a^2}$$

where

$$\eta = \frac{3800\ Z_{eff}\ \ell n\ \Lambda}{\gamma_E\ T^{3/2}} \quad \Omega\text{-cm}$$

is the Spitzer resistivity for electron temperature, T in Kelvins, $\ell n\ \Lambda$ being the Coulomb logarithm and γ_E a factor of order unity which depends only on the effective ionization state, Z_{eff}. For an element of atomic number, Z, the effective ionization state is approximately,

$$Z_{eff} = 26 \sqrt{\frac{T_{kev}}{1 + \left(\frac{26}{Z}\right)^2 T_{kev}}} \quad ,$$

with T_{kev} = electron temperature in keV. The time derivative of the inductance also acts as a resistance, given by

$$\dot{L}_p = -2\ell \frac{N-1}{N} \frac{\dot{r}}{r} \quad .$$

The current, I_p, flowing in the array then satisfies a differential equation,

$$\frac{dI_p}{dt} = \frac{V(t)-(Z_o+R_p+\dot{L}_p)I_p}{L_D+L_p} \quad ,$$

and the current flowing in each wire is $I_p(t)/N$.

The motion of the array is given by the $\underline{J}$ x $\underline{B}$ force for each wire. If each wire has mass/length = μ, the radius of the array is given by

$$\frac{d^2r}{dt^2} = - \frac{N-1}{N^2} \frac{I_p^2(t)}{\mu c^2} \frac{1}{r} \quad ,$$

which is integrated numerically as two first-order equations together with the circuit equation, using a Runge-Kutta integrator.

The radiated power is modeled as a black-body with emissivity given by

$$\varepsilon = \varepsilon^{>} f^{>} + \varepsilon^{<} f^{<} \quad ,$$

where $\varepsilon^{>}$ ($\varepsilon^{<}$) is the emissivity for photons greater (less) than a specified cut-off energy, E_*, and $f^{>}$ ($f^{<}$) is the fraction of the blackbody output which is emitted above (below) E_*. Clearly, $f^{>} + f^{<} = 1$. The total radiated energy, w_{rad}, is then given by

$$\frac{dw_{rad}}{dt} = \varepsilon\sigma T^4 A_s$$

and the yield, $w^{>}_{rad}$, above E_* is given by

$$\frac{dw^{>}_{rad}}{dt} = f^{>}\varepsilon^{>}\sigma T^4 A_s \quad ,$$

where σ is the Stefan-Boltzmann constant and $A_s = 2\pi a\ell N$ is the total surface area of the array.

To complete the description of the model, a prescription for determining the plasma temperature, T, and the wire radius, a, is required. If each wire is assumed to be a Bennett equilibruim, the Bennett pinch condition,

$$\frac{B^2}{8\pi} = n(1 + Z_{eff})\, K_B T \quad ,$$

provides a relationship between temperature and current,

$$(1 + Z_{eff})T = \frac{1}{200}\ \frac{M}{\mu}\ \frac{I_p^2}{N^2}\ \frac{1}{K_B} \quad ,$$

where M is the atomic mass and K_B is Boltzmann's constant. The wire radius, a, may be obtained from the energy balance between Ohmic dissipation and radiation,

$$I_p^2 R_p = \varepsilon \sigma T^4 A_s \quad ,$$

or

$$a = \left[\frac{\eta I_p^2 \ (10^7)}{2\pi^2 N^2 \varepsilon \sigma T^4} \right]^{1/3}$$

The model described above can be integrated until the wires just touch, $a = r \sin(\pi/N)$, at which point the system of individual wires coalesces into a plasma annulus, which rapidly assembles on axis converting the kinetic energy of implosion to temperature, radiation and outgoing kinetic energy.

For simple scaling law studies, the following very crude model has been implemented to model the assembly. The plasma annulus is converted to a cylinder of radius, r_o, as shown in Figure 2-3, with

$$r_o = a\left(1 + \frac{1}{\sin(\pi/N)}\right) \quad ,$$

where a is the wire radius when the wires just touch. The plasma temperature is adjusted so that the kinetic energy is entirely absorbed into temperature,

$$(1 + Z_{eff})\ \Delta T = \tfrac{1}{3}\ MV_r^2/K_B \quad .$$

The system is then allowed to radiate and cool for a period of five MHD growth times, calculated as Alfven transit times,

$$\tau = 5(r_o/V_A),$$

where $V_A = (B^2/4\pi\rho)^{\frac{1}{2}}$ is the Alfven speed, B is the magnetic field at $r = r_o$ due to the current I_p and ρ is the mass density, $\rho = N\mu/\pi r_o^2$. During the cooling period, assuming blackbody radiation, the code separately integrates for the total yield and the yield above E_*.

A test case has been run for an Aluminum (Z=13, A=27) array driven at constant voltage, $V(t)=V_o$, with the following parameters:

N = Number of Wires = 6

$N\mu\ell$ = Array Mass = 100 μg

ℓ = Array Length = 3 cm

r(o) = Initial Array Radius = 2.2 cm

b = Return-Current Radius = 3 cm

V_o = Open-Circuit Voltage = 3 MV

Z_o = Generator Impedance = 0.7 Ω

L_D = Diode-Housing Inductance = 10 nH

$\varepsilon^{>}$ = Emissivity for $h\nu$>1kev = 5×10^{-6}

$\varepsilon^{<}$ = Emissivity for $h\nu$<1kev = 5×10^{-4}

The characteristics of the implosion are summarized in Figures 2-4 thru 2-7. As seen in Figure 2-4, the implosion of this array requires approximately 69 ns. At the time the wires touch, as in Figure 2-3, the wires have achieved an inward speed of 1.3×10^8 cm/sec. The individual wire radius, a, varies over a factor of two during most of the implosion. At very early times, this radius is artifically large due to the assumptions of constant emissivities and the prescription of choosing "a" as the radius where Ohmic heating is balanced by radiative cooling. Experiments at Maxwell[2] have displayed an initial pinching of the individual wires followed by an expansion of the wires, which is at least qualitatively as shown in these calculations.

Figure 2-5 shows the temperature and average ionization state vs. time for this implosion. The temperature, which is tied to the current by the Bennett pinch condition, peaks at approximately 655 eV at peak current, and subsequently drops to 249 eV by the end of the run-in. The ionization state, Z_{eff}, is between 9 and 11 during most of the implosion.

The circuit equation may be converted to a power equation by multiplying both sides by the total current, I_p, to obtain

$$I_p V = \frac{d}{dt}\left[\frac{1}{2}(L_D+L_p)I_p^2\right] + (R_o+R_p)I_p^2 + \frac{1}{2}I_p^2\dot{L}_p \quad ,$$

where $I_p V$ is the input power from the external generator, which must equal the rate at which energy is stored in the magnetic field, Ohmic power losses, and the rate at which energy is stored as kinetic energy of the array (the $\dot{L}_p$ term). Figure 2-6 illustrates how these various components of the power equation vary in time. The rate at which energy is stored in the magnetic field is not plotted, but is just the difference between the $V_o I_p$ curve and the sum of the other two curves. At the end of the run-in, the wires are acquiring kinetic energy at a rate which exceeds $V_o I_p$, and in fact the implosion is tapping stored field energy just prior to assembly on axis.

Figure 2-7 shows the evolution of the various energy channels during the implosion. During run-in, when the temperature is low, internal energy and radiation are relatively small compared with field energy and kinetic energy. Again, at the end of the run-in, the field energy decreases rapidly as the kinetic energy increases.

After the wires touch, at t = 69 ns, the code instantly converts the annulus to a cylinder and "shock heats" it by converting all the kinetic energy to internal energy. This prescription in the test case yields a cylinder of 0.7 cm diameter with an ion density of 2×10^{20} cm^{-3} at a temperature of 11.4 keV. The cylinder cools rapidly by radiative power loss, and after five Alfven transit times (or 5.1 ns) its temperature has dropped to 113 eV. In this problem, the total energy radiated at all frequencies is 34.2 kJ and the energy above 1 keV is 6.7 kJ, assuming a blackbody spectrum. During the run-in, the individual wires radiated 28.2 kJ at all frequencies, but only 1.7 kJ above 1 keV. The radiation above 1 keV, therefore, occurs after collapse in this model, but a significant fraction of the total yield can occur during the run-in. The model assumes, of course, that the individual wires remain stable and do not develop "hot spots" during the collapse. If hot spots develop, they may cause a larger fraction of the yield above 1 keV to occur during the run-in than is calculated here.

Treating the test case described above as a base case, a parameter study has been made to test the sensitivity of the radiation yield to variations of several of the input parameters. The results of this study are shown in

Table 2-1 and in Figures 2-8 thru 2-15. Multiple parameter variations from the base case have not been attempted, but rather only a single parameter has been altered for each of these runs. In each case the total radiation yield, W_T, and the yield $W^>$, for $h\nu > 1$ keV, is presented. These are calculated as described above, using a blackbody spectrum with different emissivities above and below 1 keV, and including the radiation from a collapsed, "shock-heated" plasma cylinder as it cools during five Alfven transit times.

Figure 2-8 shows the variation with N, keeping the total array mass fixed at 100 μg. As the number of wires increases, the time for impact is shortened, thereby reducing the total yield by more than two-fold. The yield above 1 keV, however, becomes very insensitive to the number of wires for N>12.

Figure 2-9 shows the effect of varying the total array mass, with the number of wires fixed at N=6. Here, the heavier arrays are worse as expected in this model. Increasing the initial array radius, r(o), as shown in Figure 2-10, improves the yield. The larger arrays (at 100 μg) take longer to collapse, acquiring more kinetic energy. Also, since the current return has been fixed at b = 3 cm, the initial inductance, L_p, for the larger arrays is smaller, thereby slightly reducing the current risetime.

Figures 2-11 thru 2-13 show the effect of variations of the external circuit parameters. Increasing the open-circuit voltage, Figure 2-11, or reducing the generator impedance, Figure 2-12, both strongly improve the yield. While this trend suggests that the yield may simply depend monotonically on the power, V_o^2/Z_o, independent of whether V_o or Z_o is the quantity varied, a detailed look at Table 2-1 shows that this proposition is not correct. The yield for Z_o = 1.5 Ω and V_o = 3MV (V_o^2/Z_o = 6 TW) is significantly lower than the yield for Z_o = 0.7 Ω and V_o = 2MV (V_o^2/Z_o = 5.7 TW).

Figure 2-13 shows that the yield is insensitive to the diode-housing inductance in the range 10 nH to 20 nH. This insensitivity is probably due to the plasma inductance, which varies from nominally 5 nH to 30 nH during the implosion.

This model has been utilized to provide initial conditions for the SQUEEZE code, which computes the collapse of a plasma annulus. The WIRES model, described above, is run until the individual wires just touch. A bridge subroutine is then employed to convert the wire array to an imploding plasma annulus. The wire array, consisting of N wires of radius, a_f, on a circle of radius, r_f, is converted to an annulus with outer radius, r_o, and inner radius, r_i, given by

$$r_o^2 - r_i^2 = Na_f^2$$

$$r_i r_o = r_f^2 \quad .$$

The SQUEEZE code then continues the calculation, using a more sophisticated radiation and hydrodynamics model.

REFERENCES

1. D. Mosher, NRL Memorandum Report 3687 (1978); J.J. Katzenstein, J. Appl. Phys. 52, 676 (1981); H.W. Bloomberg, J. Appl. Phys. 51, 202 (1980).

2. W. Clark, E. Chu, M. Gersten, J. Katzenstein, A. Kolb, A. Miller, J. Pearlman, A. Peratt, J. Rauch, R. Richardson, J. Riordan, J. Shannon, and M. Wilkinson, Proc. 4th Int'l Top. Conf. on High-Power Electron and Ion-Beam Research and Technology (Polaisean, France, 1981), Vol. I, P. 279.

TABLE 2-1

A PARAMETER STUDY USING THE "WIRES" CODE*

Number of Wires	6	12	24										
Total Array Mass (μg)	100			200	300								
Initial Radius (cm)	2.2											2.0	2.5
Open-Circuit Voltage (MV)	3							2	4				
Generator Impedance (Ω)	0.7					1.0	1.5						
Diode-Housing Inductance (nH)	10									15	20		
Yield Above 1 kev (kJ)	6.7	4.8	4.0	5.1	4.2	3.1	1.0	2.2	14.3	6.6	6.4	6.1	7.6
Total Radiation Yield (kJ)	34.2	24.5	15.5	31.7	28.8	24.8	14.1	21.0	46.0	33.2	32.2	30.7	39.4

* All parameters which are unspecified have the values specified for them in the first column, which is the base case.

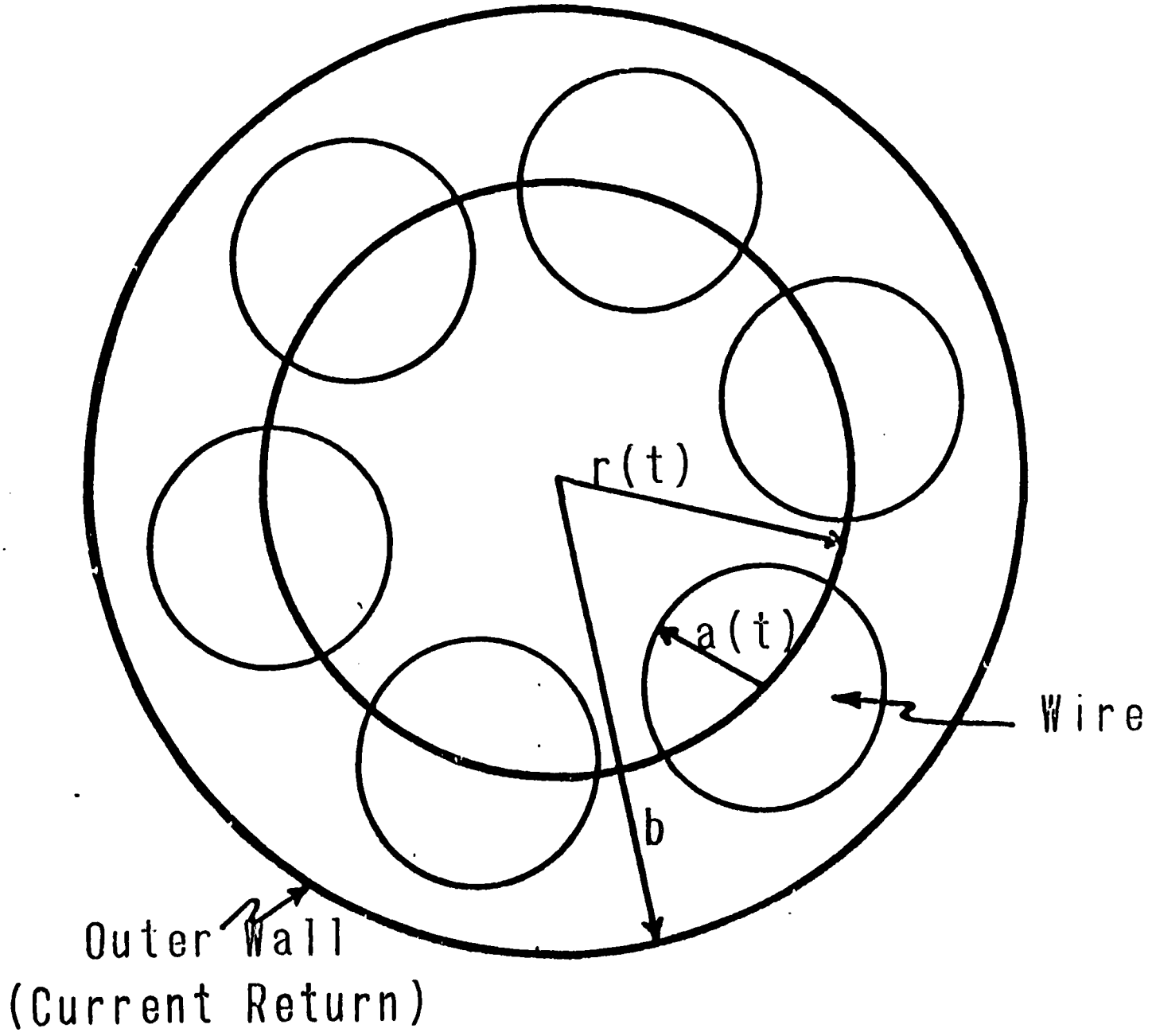


Figure 2-1. Schematic Representation of the Wire Array Model

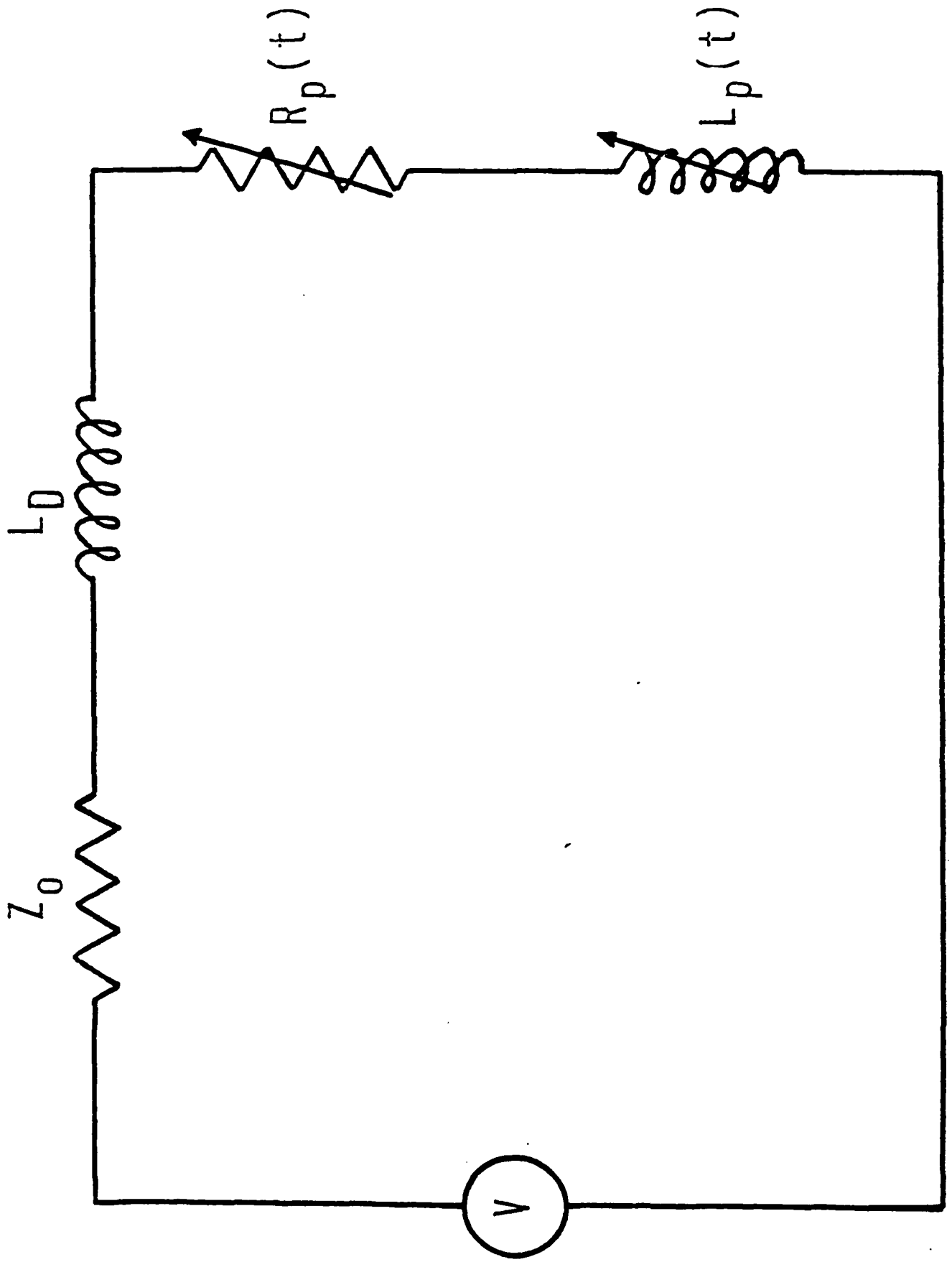


Figure 2-2. External Circuit for the Wire Array Model

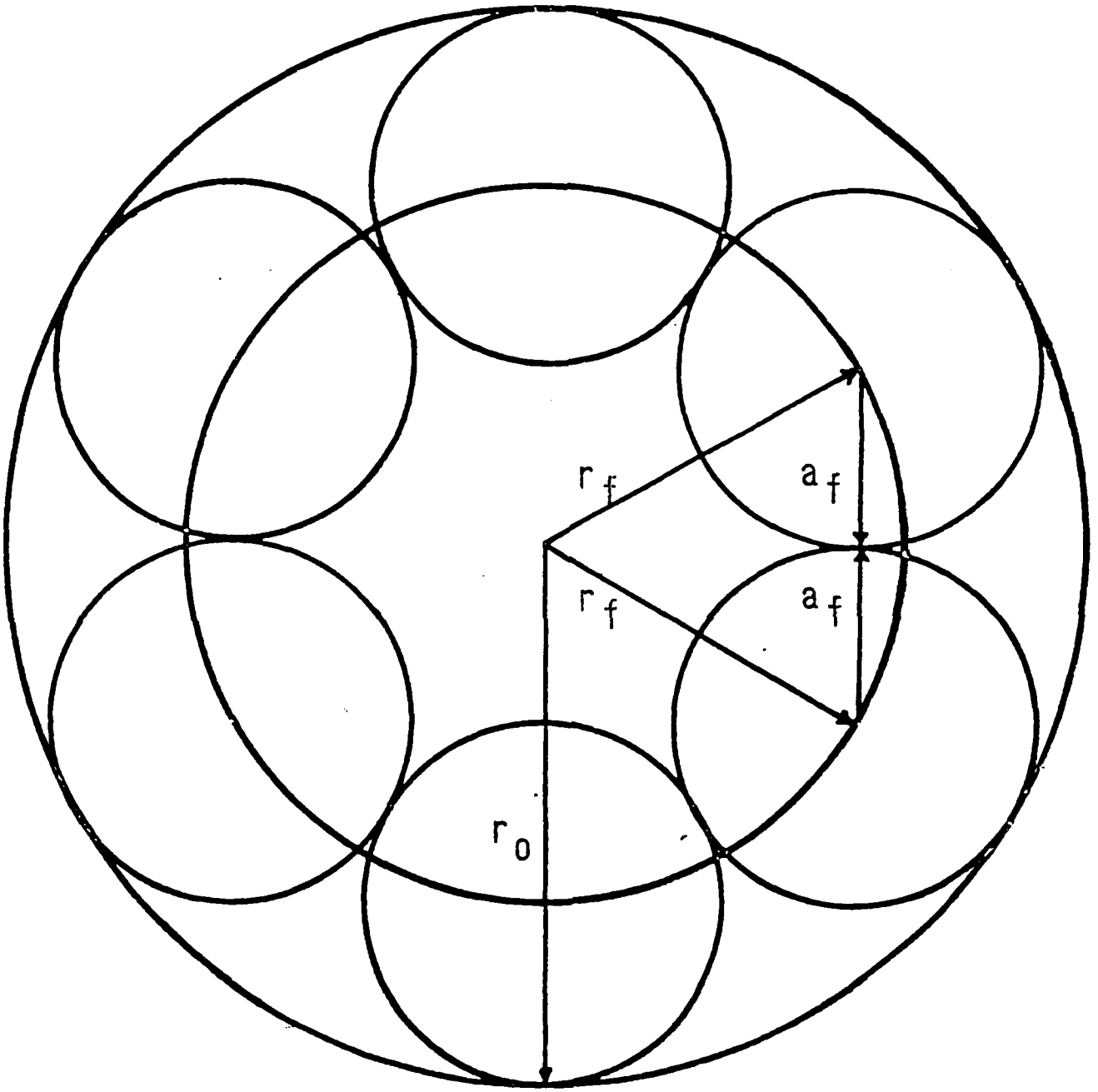


Figure 2-3. Conversion of Wire Array to Plasma Cylinder

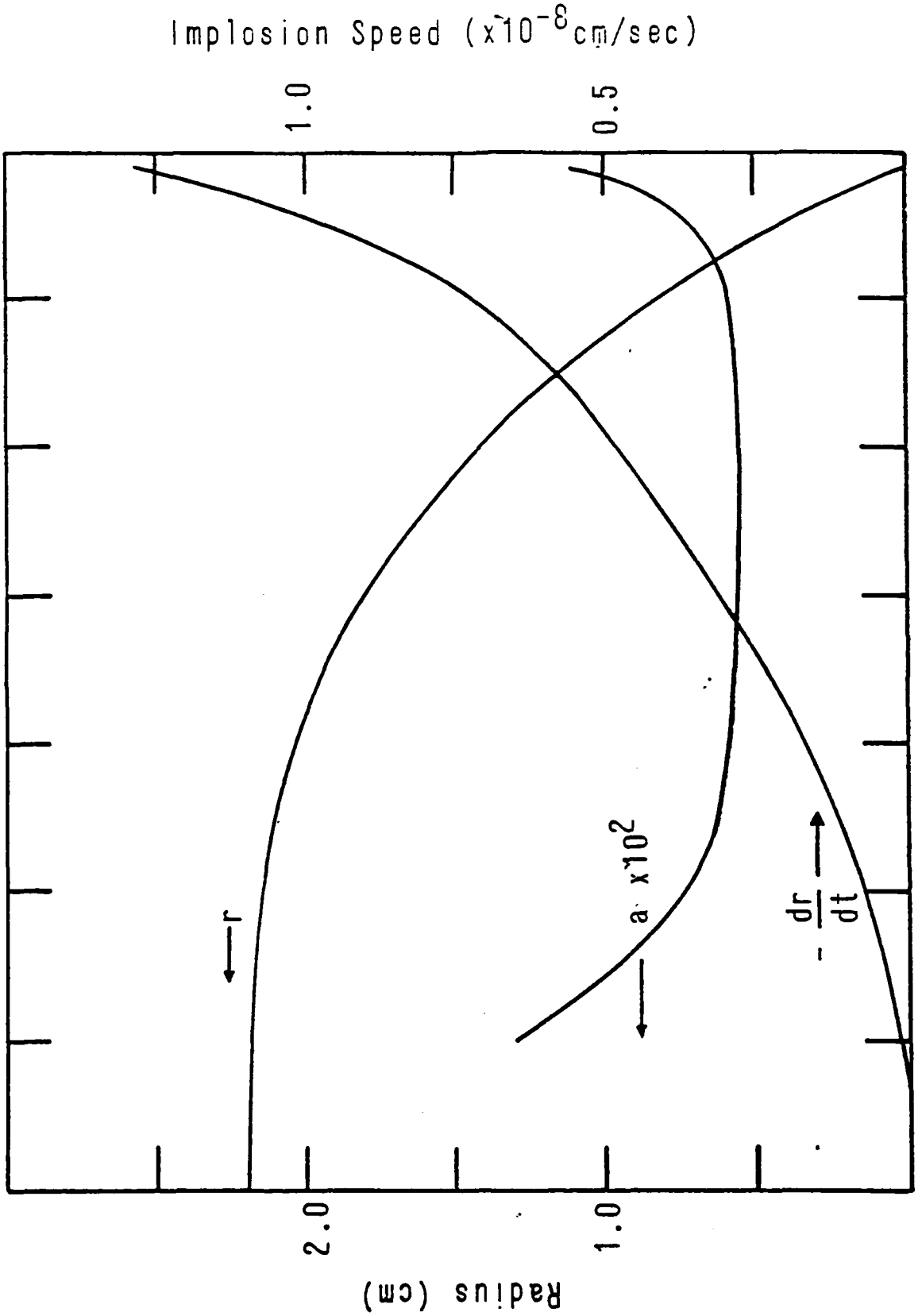


Figure 2-4. Array Radius, Individual-Wire Radius and Implosion Speed vs. Time for the Base-Case

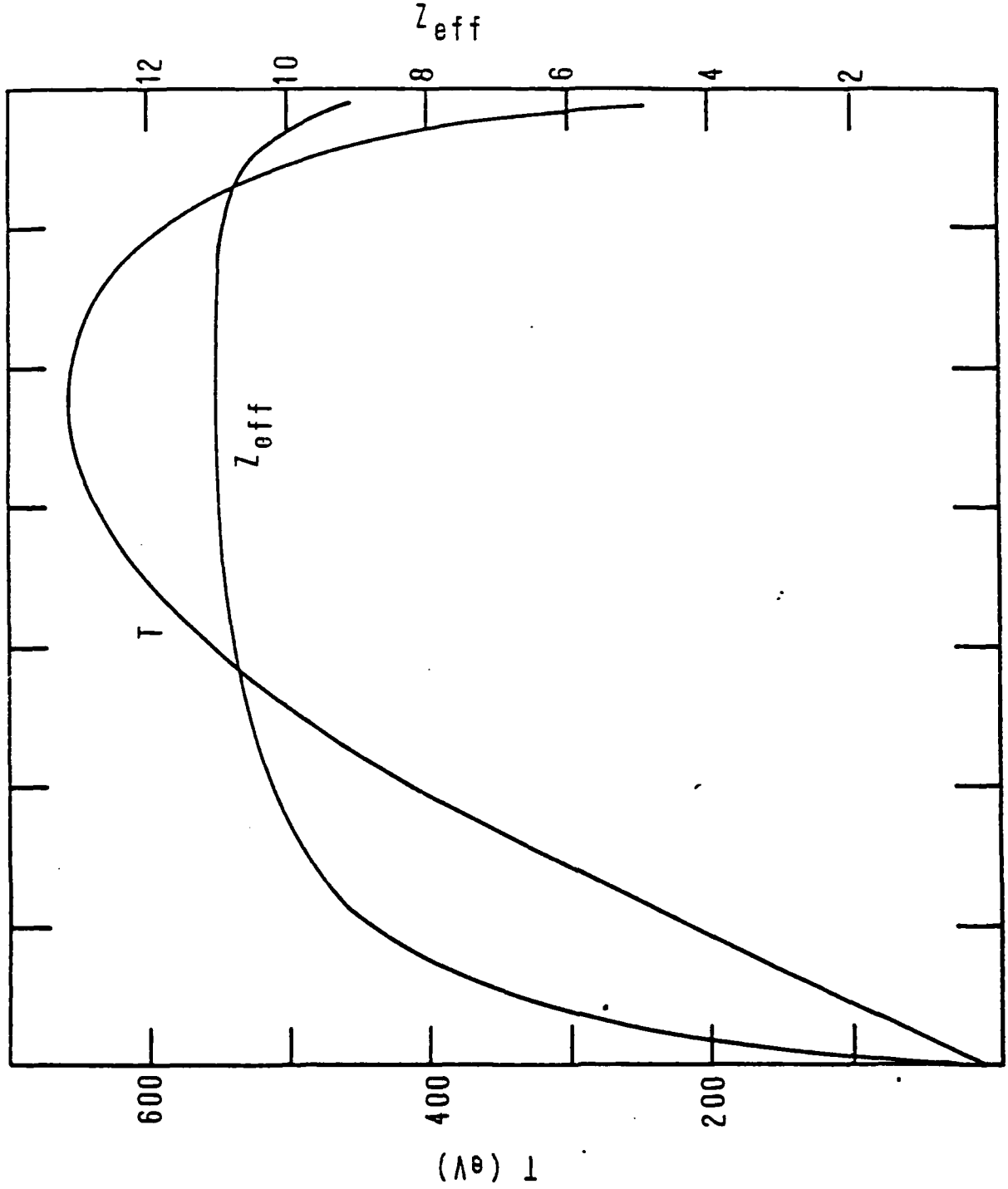


Figure 2-5. Temperature and Average Ionization Level vs. Time for the Base Case

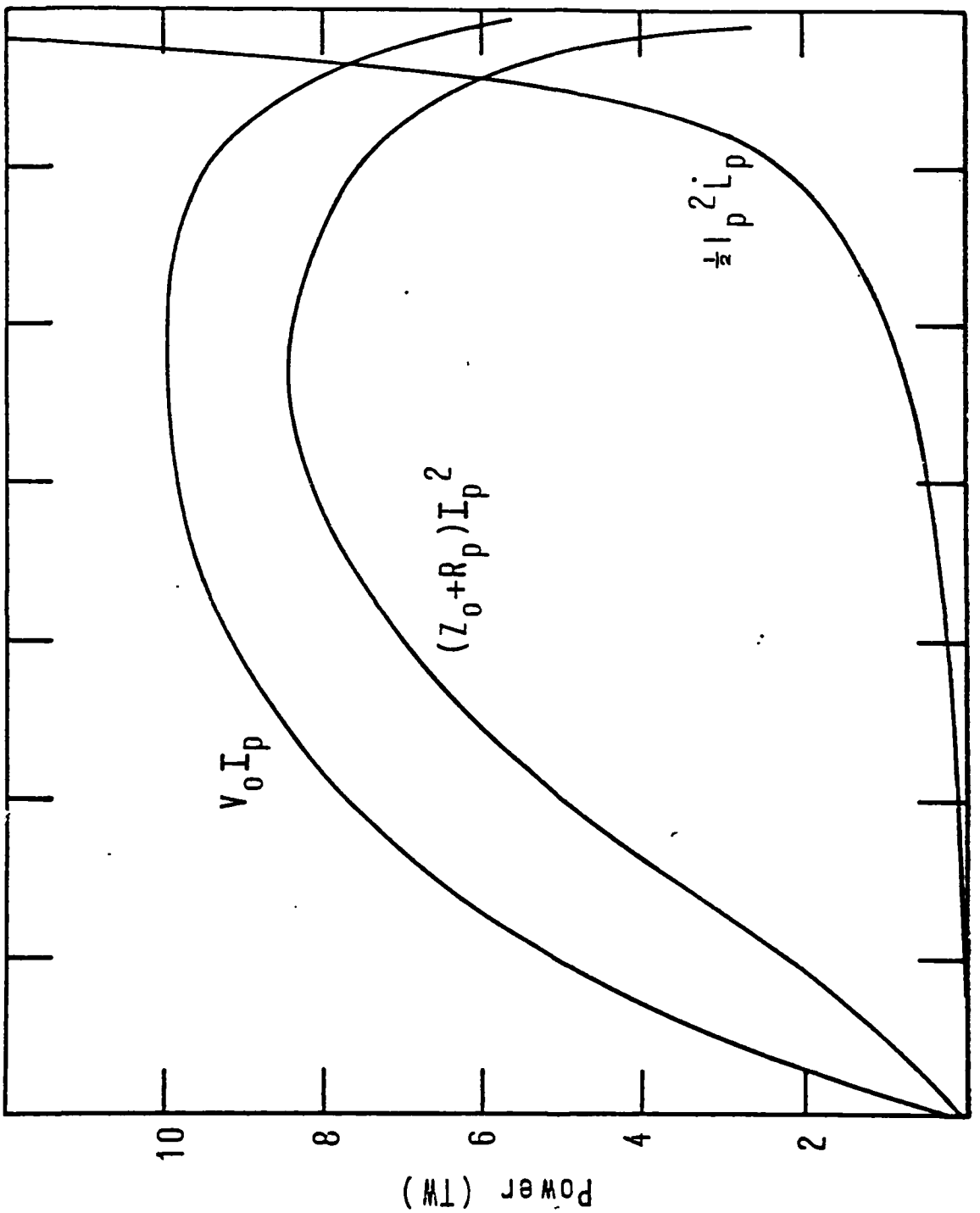


Figure 2-6. Input Power, Ohmic Power, and Power to Kinetic Energy vs. Time for the Base Case

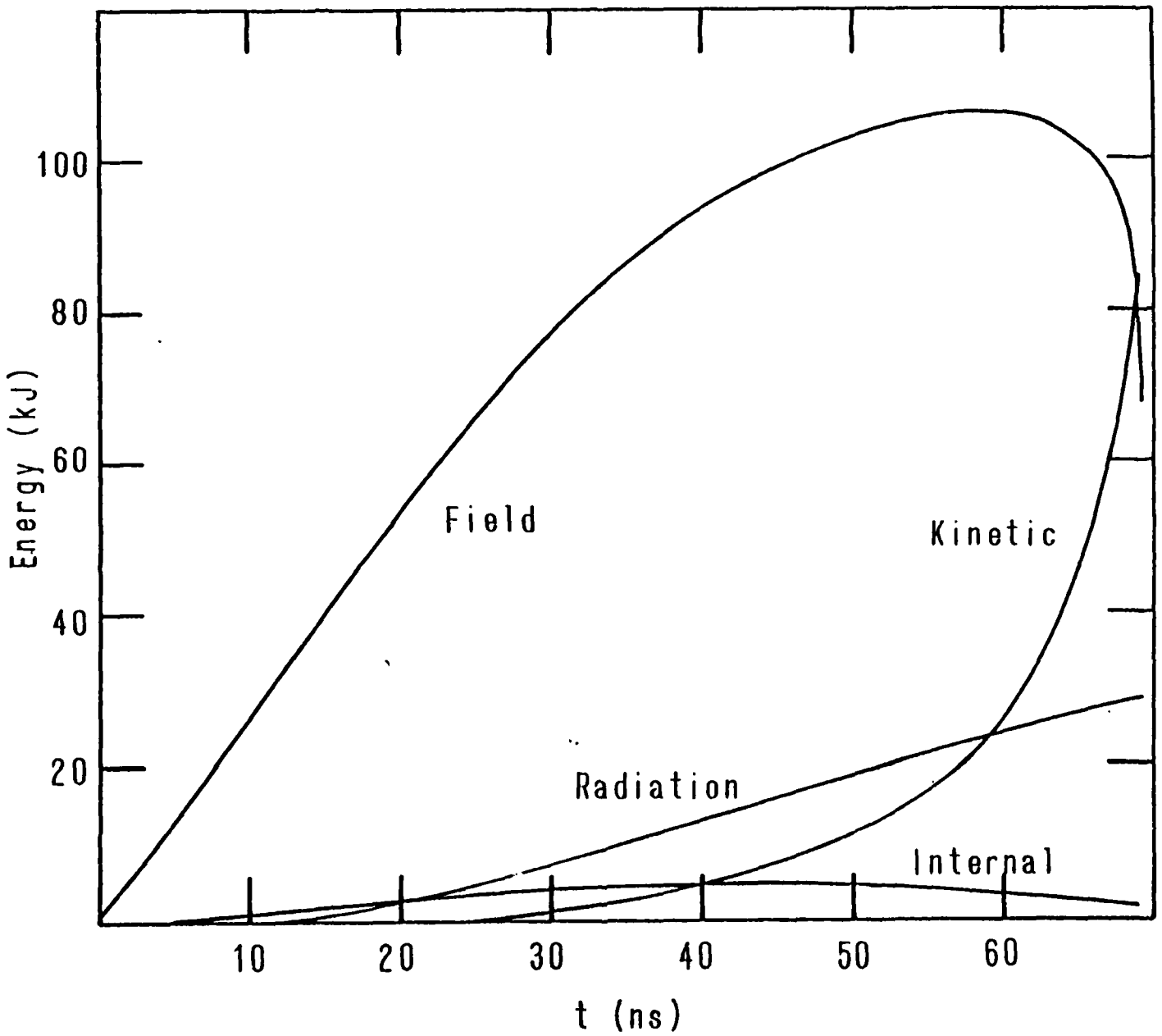


Figure 2-7. Energy Channels vs. Time for the Base Case

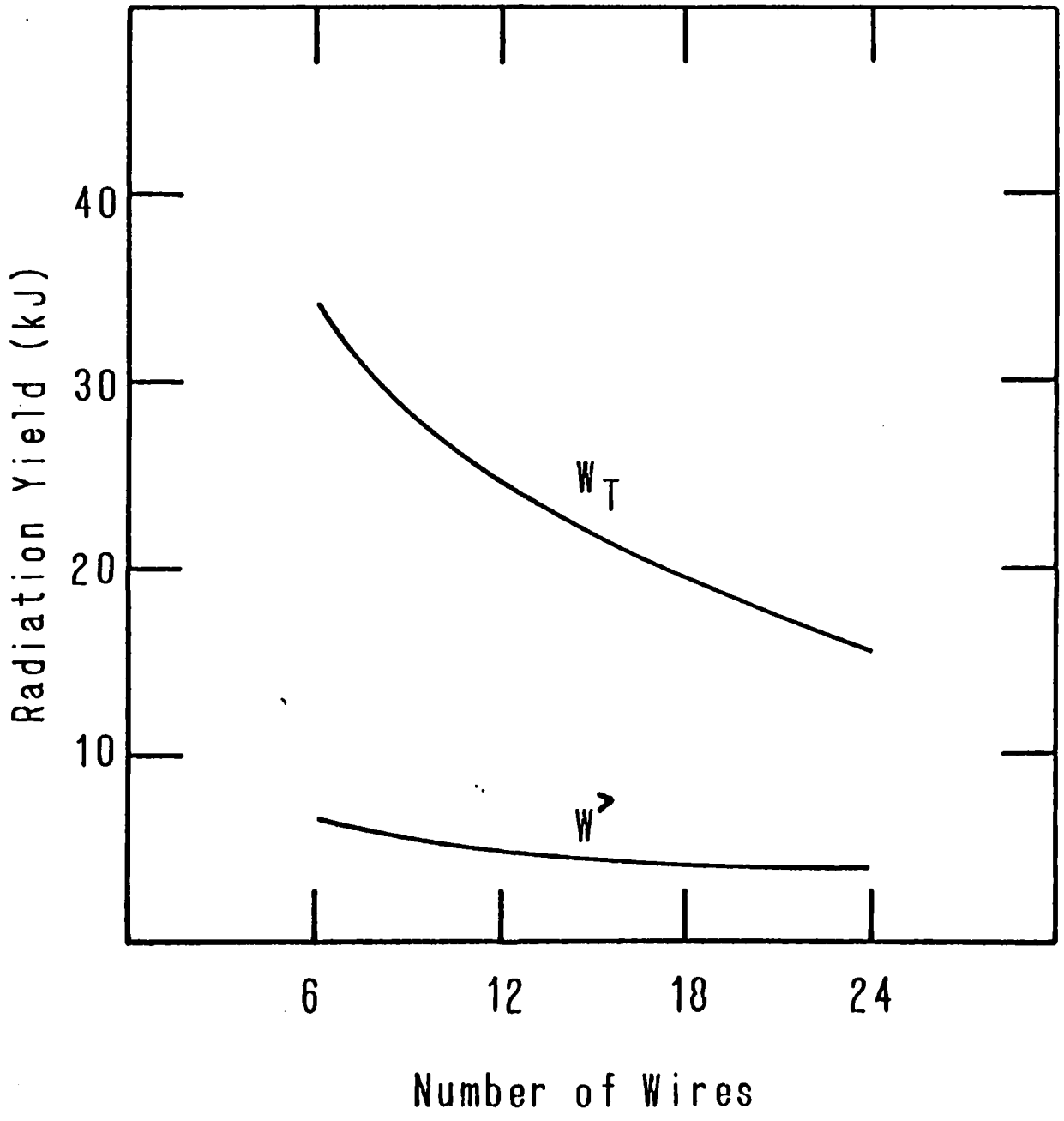


Figure 2-8. Total Radiation Yield and Yield Above 1keV vs. Number of Wires

Figure 2-9. Total Radiation Yield and Yiel .bove 1keV vs. Total Array Mass

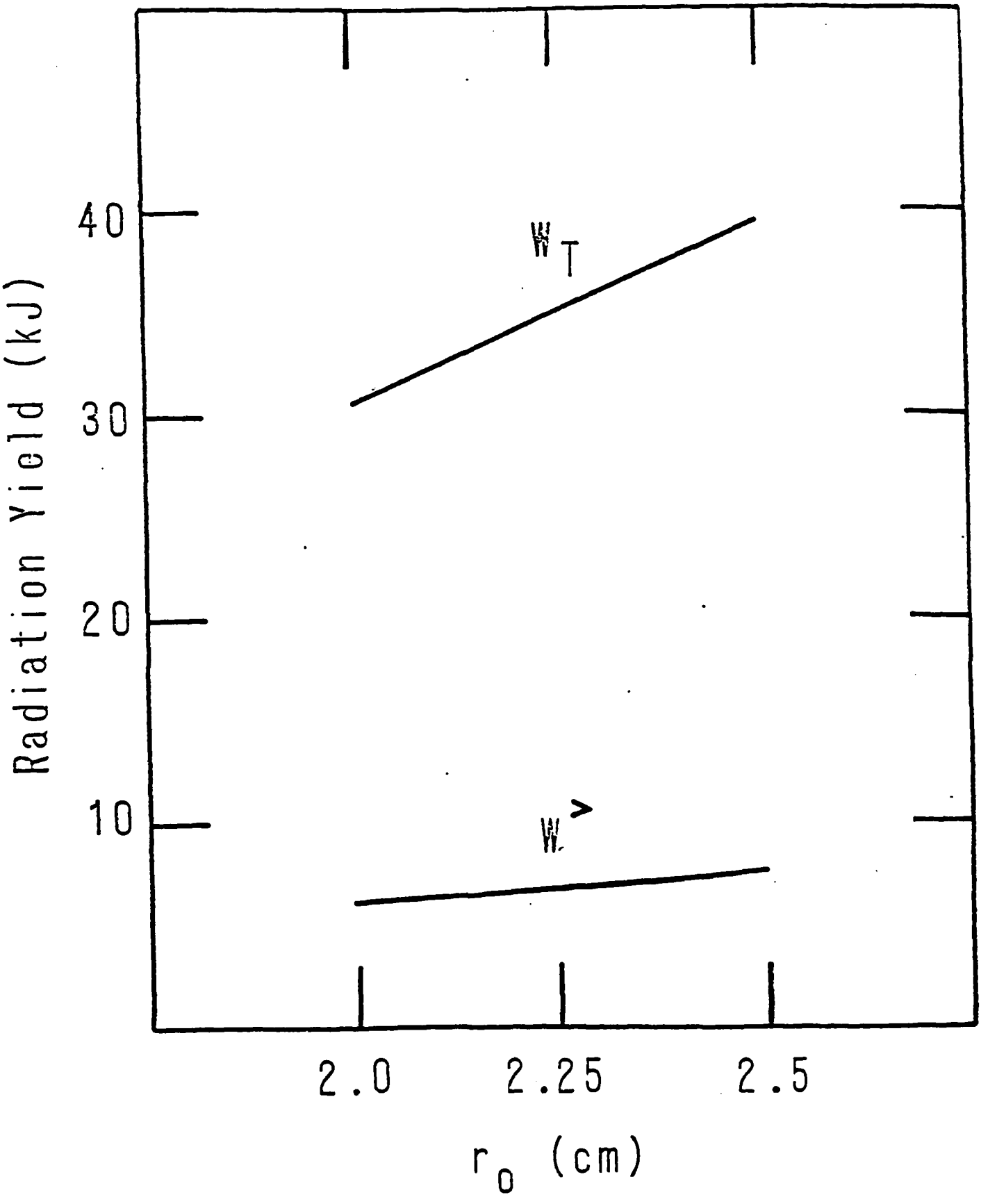


Figure 2-10. Total Radiation Yield and Yield Above 1keV vs. Initial Array Radius

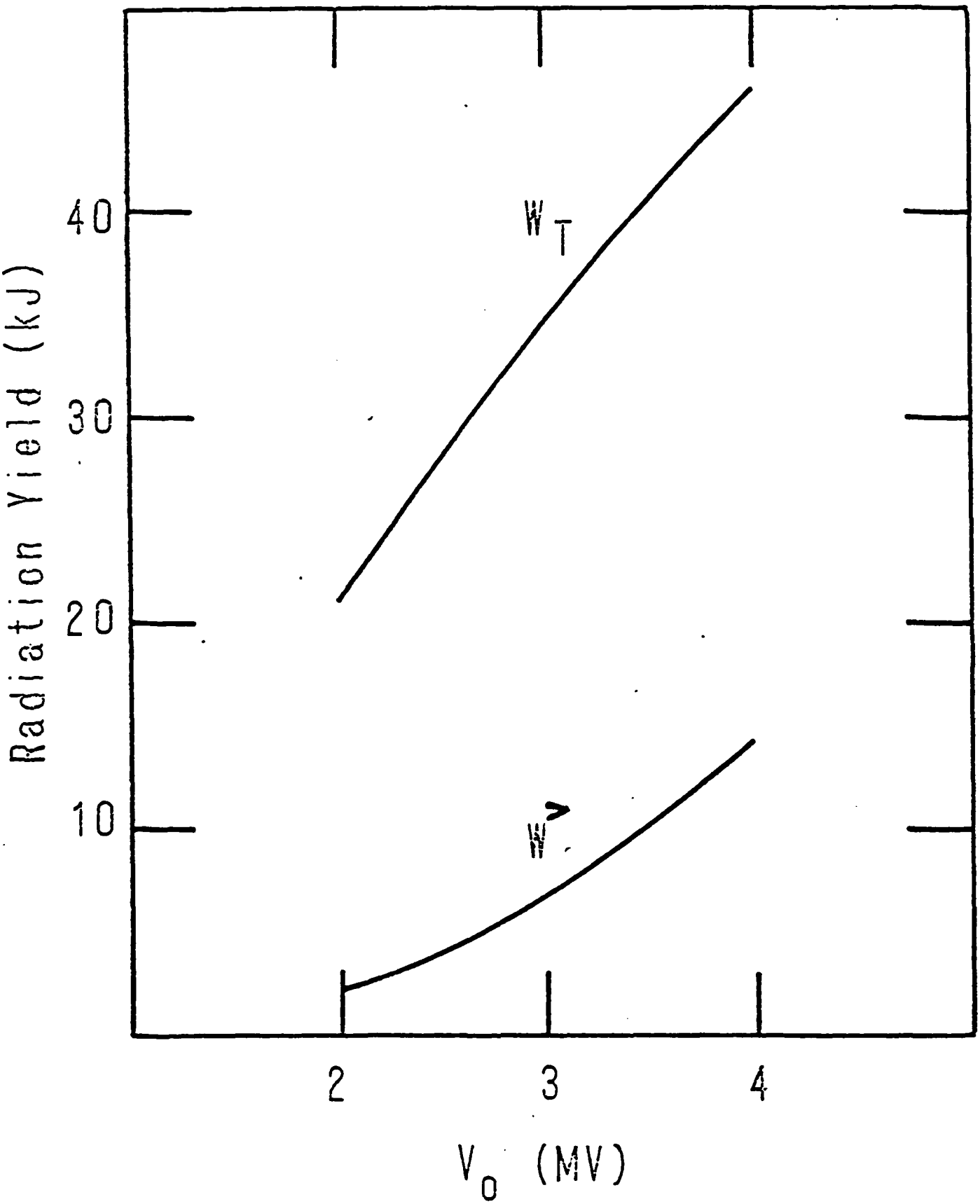


Figure 2-11. Total Radiation Yield and Yield Above 1keV vs. Open Circuit Voltage

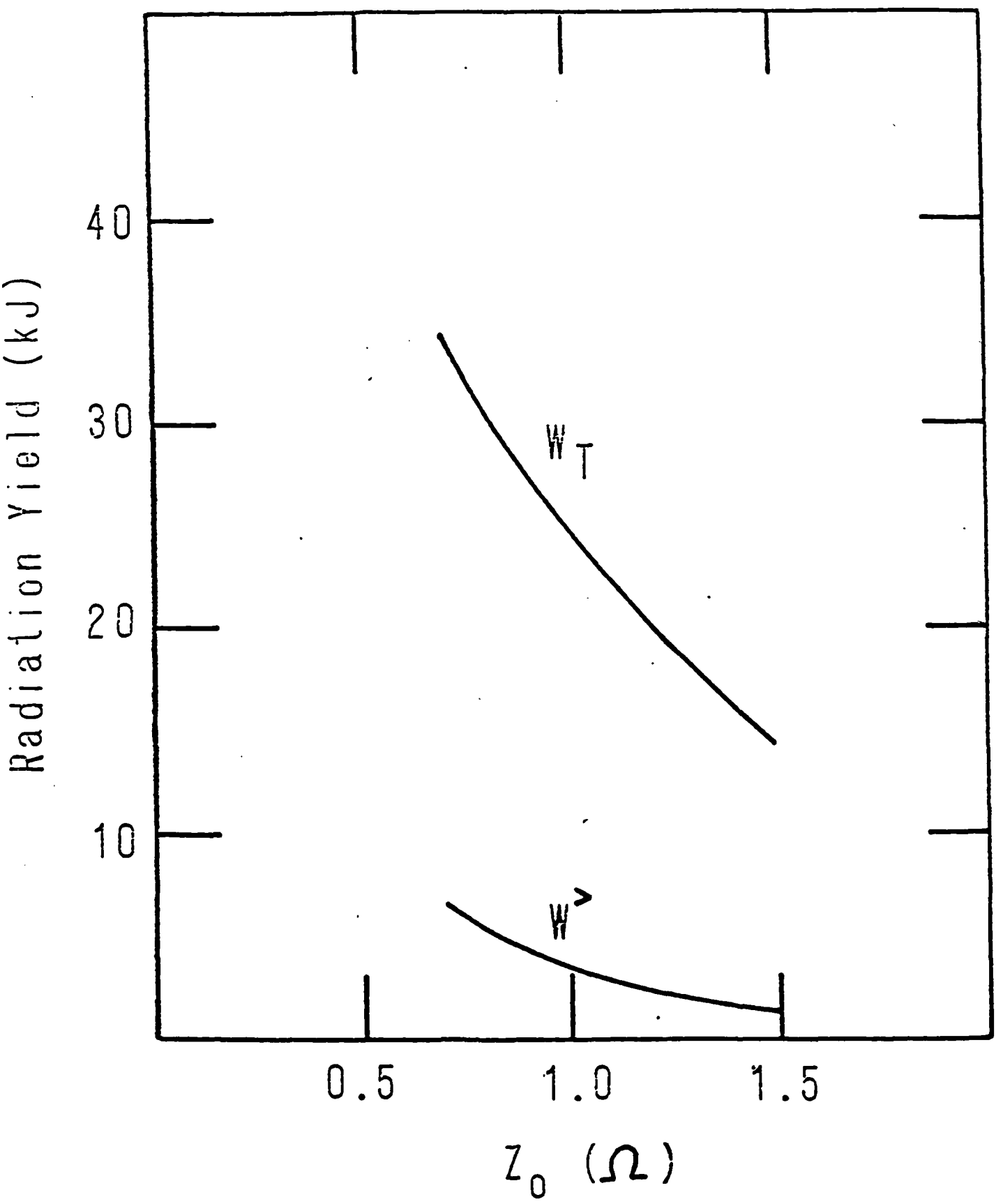


Figure 2-12. Total Radiation Yield and Yield Above 1keV vs. Generator Impedance

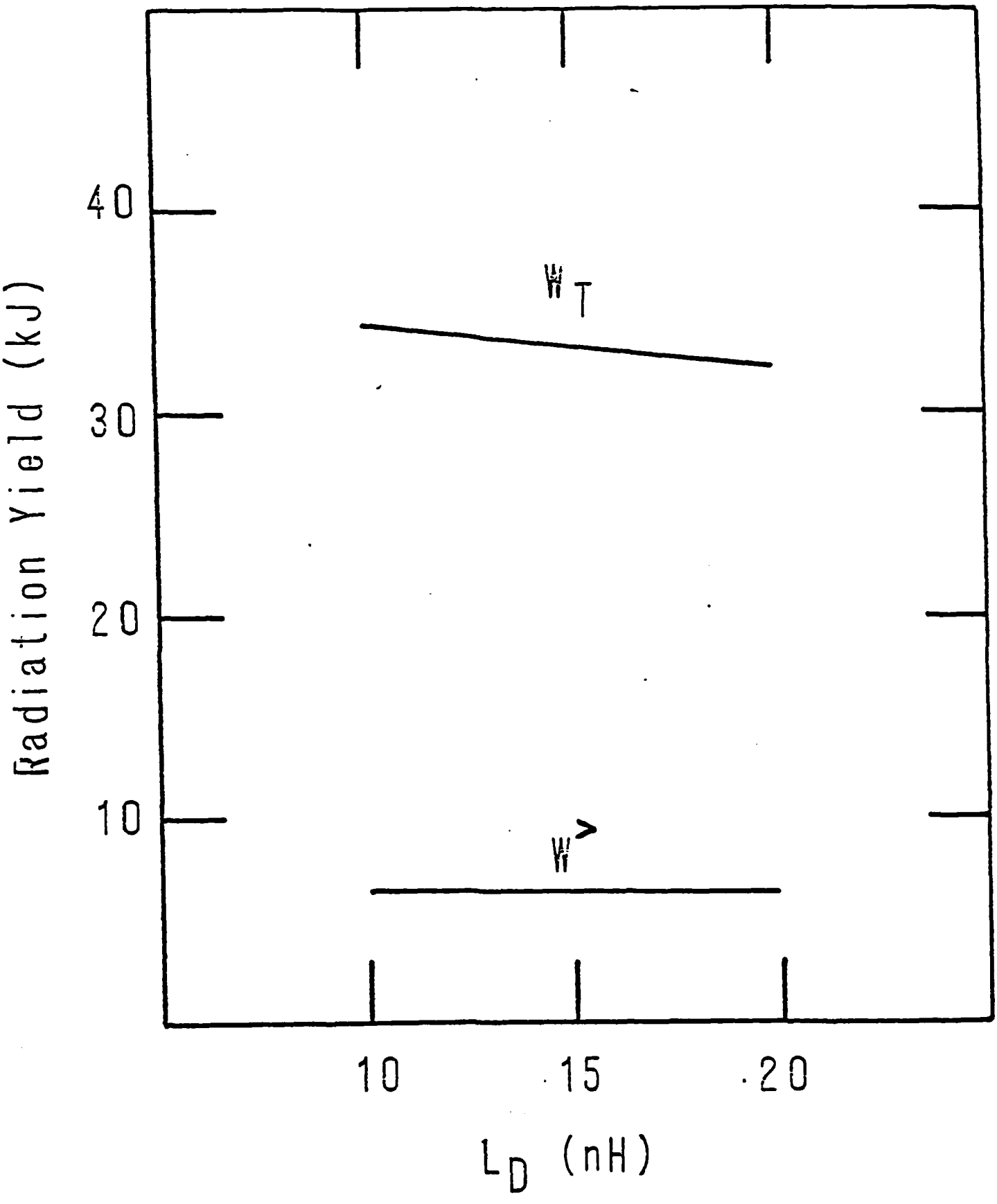


Figure 2-13. Total Radiation Yield and Yield Above 1keV vs. Diode-Housing Inductance

SECTION 3

LINEAR, IDEAL MHD STABILITY ANALYSIS

Experiments on imploding wire arrays, gas puffs and foils have displayed hot spots, beads, plasma jets and kinks, all of which are believed to be manifestations of MHD instabilities. These phenomena couple strongly to the plasma dynamics and may actually determine the strength of the pinch and the time duration of the assembled plasma. The densities and high temperatures of the hot spots or beads may provide the conditions needed for generating most of the radiation above 1 keV.

The usual simple test for the importance of MHD instabilities in a plasma system is whether the time required for an Alfven wave to cross the system is short compared with the confinement time of the system. The Alfven speed is given by

$$v_A = (B^2/4\pi\rho)^{1/2} \quad ,$$

where B is the magnetic field and ρ is the mass density. To make this argument specific, assume that a wire array with a mass of 100 μg has collapsed to a plasma cylinder of 0.1 cm radius and 3 cm length, and is carrying a current of 2 MA. For these parameters, which are typical of experimental conditions, the mass density is $\rho = 10^{-3} g/cm^3$ and the magnetic

field at the edge of the cylinder is B = 4 MG, which yields $v_A = 3.6 \times 10^7$ cm/sec and the Alfven transit time across the plasma radius is 2.8 ns, which is only about 10% of the observed radiation pulsewidth and the observed plasma confinement time. MHD instabilities are therefore expected to be important in this system.

During the implosion, currents penetrate into the plasma wires and/or annulus, and the MHD growth rate will be sensitive to the actual current distribution in the plasma. The radiation-coupled hydro codes, WHYRAD and SPLATT, model the field penetration and can provide an "equilibrium" configuration for the assembled plasma. During the run-in phase of the implosion, inertial terms in the zero-order equations will be important. A formulation for the MHD instability growth rate for cylindrical MHD equilibria is described in this section for an arbitrary equilibrium current distribution which is consistent with equlibrium pressure balance. Several references[1-3] discuss this type of model.

The linearized equations for ideal MHD may be written in terms of density ρ, velocity $\underline{v}$, pressure p, current density $\underline{J}$, and magnetic field $\underline{B}$ as

$$\rho^o \frac{\partial \underline{v}^1}{\partial t} = \nabla p^1 + \underline{J}^o \times \underline{B}^1 + \underline{J}^1 \times \underline{B}^0$$

$$\nabla \times \underline{B} = \frac{4\pi}{c} \underline{J}$$

$$\frac{1}{c} \frac{\partial \underline{B}^1}{\partial t} = \nabla \times (\underline{v}^1 \times \underline{B}^o)$$

$$\frac{\partial p^1}{\partial t} = -\underline{v}^1 \cdot \nabla p^o - \Gamma p^o \nabla \cdot \underline{v}^1 \quad ,$$

where superscripts "o" and "1" denote zero-order and first-order quantities, respectively, and Γ is the ratio of specific heats.

These equations may be expressed as a second-order equation for the displacement vector, $\underline{\xi}(\underline{x},t)$, defined as

$$\underline{\xi}(\underline{x},t) = \int_o^t \underline{v}^1(\underline{x},t') \, dt' \quad ,$$

to obtain

$$\rho^o \frac{\partial^2 \underline{\xi}}{\partial t^2} = \underline{F} \{\underline{\xi}\}$$

$$= \nabla(\underline{\xi} \cdot \nabla p^o + \Gamma p^o \nabla \cdot \underline{\xi}) + \frac{1}{4\pi} (\nabla \times \underline{B}^o) \times [\nabla \times (\underline{\xi} \times \underline{B}^o)]$$

$$+ \frac{1}{4\pi} (\nabla \times [\nabla \times (\underline{\xi} \times \underline{B}^o)]) \times \underline{B}^o$$

This formula is the starting place for all linear, ideal MHD stability analyses.

For a circular cylinder equilibrium, the coefficients in the linearized MHD equations are independent of θ and z. Each Fourier harmonic of the perturbation will therefore evolve independently, and the perturbation may be expressed as

$$\underline{\xi}(\underline{x},t) = \underline{\xi}(r)e^{i(kz+m\theta-\omega t)} \quad .$$

In this case it has been shown[1] that the problem reduces to a single, homogeneous, second-order equation of the eigenfunctions associated with the radial displacement, ξ_r. This equation is given by

$$(\alpha\xi_r')' + q\xi_r = 0 \quad ,$$

where prime indicates differentiation with respect to the radial coordinate. The coefficients, α and q, are given by

$$\alpha = \frac{rAC}{\lambda_{12}}$$

$$q = r\left[\frac{\det\Lambda}{AC\lambda_{12}} + \left(\frac{\lambda_{11}}{\lambda_{12}}\right)'\right] \quad ,$$

where Λ is a 2 x 2 matrix of the form,

$$\Lambda = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ rAC\left(\frac{1}{r}\, p_*'\right)' + \tilde{\lambda}_{21} & \frac{AC}{r} - \lambda_{11} \end{pmatrix} .$$

For equilibria with no flow and without an axial magnetic field, i.e. $B = B_\theta$, these quantities may be expressed in terms of the Alfven speed, v_A, and the sound speed, c_s, where

$$v_A^2 = B_\theta^2/4\pi\rho$$

$$c_s^2 = \Gamma p/\rho$$

as

$$\lambda_{11} = \frac{AC}{r} - \rho C\, \frac{2mv_A^2}{r^3} - \frac{2}{r}\, \rho^3\omega^4 v_A^2$$

$$\lambda_{12} = \rho^2\omega^4 + C\, \left(k^2 + \frac{m^2}{r^2}\right)$$

$$\tilde{\lambda}_{21} = C\left[A^2 - \frac{4m^2\rho^2 v_A^4}{r^4}\right] - \frac{4}{r^2}\, \rho^4\omega^4 v_A^4$$

$$A = \rho\, \left[\frac{m^2}{r^2}\, v_A^2 - \omega^2\right]$$

$$C = \rho^2 C_s^2\, \left[\frac{m^2}{r^2} v_A^2 - \omega^2\right] - \rho^2\omega^2 v_A^2 \quad ,$$

and p_* denotes the total equilibrium pressure, which satisfies

$$p_*' = -\frac{\rho v_A^2}{r} \quad .$$

The functions $A, C, \lambda_{11}, \lambda_{12}$ and $\tilde{\lambda}_{21}$ are therefore algebraic functions of the eigenvalue parameter, ω^2, and the equilibrium fields.

The matrix, A, contains the second derivative of p_*, and the coefficient, q, requires the derivative of $\lambda_{11}/\lambda_{12}$. These non-algebraic features can be troublesome, particularly when the equilibrium data are obtained numerically. The numerical computation of these derivatives is expected to require some smoothing of the equilibrium data.

The boundary conditions on ξ_r are that it vanish at the outer boundary, assumed to be a conducting wall, while regularity at the origin may be used to determine its behavior at r = 0 from an indicial equation. Writing $\xi_r = r^\mu \Sigma_{j=0}^{\infty} a_j r_j$, the solution near the origin satisfies $\mu^2=1$ for m=0 and $(\mu+1)^2 = m^2$ for $m\neq 0$.

With this formulation, the problem is completely posed in terms of functions of the equilibrium fields. In ideal MHD, the eigenvalue is ω^2, implying solutions that are either purely oscillatory or purely growing, a feature which follows from the self-adjoint nature of the perturbed fluid equations. More generally, eg. for equilibria with flow, the self-adjoint property is lost, and the eigenvalues will be complex.

The cylindrical MHD stability problem, as formulated above, may be solved numerically by a "shooting code".[3] In this technique one selects the equilibrium and sets ξ_r near the origin to satisfy the indicial equation for a trial value of ω^2. By solving repeatedly for $\xi_r(r)$ for different ω^2, a value is found for which $\xi_r(\text{wall}) = 0$, the outer boundary condition. This technique is a widely used approach.

In the limit of surface currents, the linear stability problem can be solved exactly and analytically. The eigenvalue is given by

$$\omega^2 = \left(\frac{v_A}{a}\right)^2 \quad [-\beta_1 ka + m^2 \beta_2] \quad ,$$

where v_A is the Alfven speed at the edge of the plasma (r=a) and a perfectly conducting wall is assumed at r=b. The coefficients, β_1 and β_2, are given by

$$\beta_1 = \frac{I_m'(ka)}{I_m(ka)}$$

$$\beta_2 = \beta_1 \frac{K_m(ka) I_m'(kb) - K_m'(kb) I_m(ka)}{K_m'(kb) I_m'(ka) - K_m'(ka) I_m'(kb)} \quad ,$$

where I_m and K_m are modified Bessel functions.

The ideal MHD linear growth rate is plotted against ka for various azimuthal mode numbers, m, and various b/a values in Figure 3-1. The results appear quite insensitive to b/a for $b/a \geq 5$. As b/a nears unity, however, the instabilities with m>0 are stabilized by wall stabilization. On the figure, an instability is indicated by a negative eigenvalue, i.e. $\omega^2 < 0$. The m=0 sausage mode is always unstable. The modes for m>0 become unstable as ka increases. The m=1 kink mode, on Figure 1, becomes stable as ka nears zero. For ka≤1, however, there is a k-band where the m=1 mode has a larger growth rate than the m=0 mode. The linear theory predicts that all the modes will become unstable for ka>>1. At very short wavelengths, however, the instability is destroyed by small-scale turbulence and mixing of the plasma. In practice, the largest growth rates are expected for ka ~ 1.

A more recent approach to the numerical solution of these problems has been developed at the University of Texas at Austin[4], and consists of a finite element technique. The equilibrium fields are developed in a representation by B-splines, which form the basis functions for the finite-element solution. The differential equation for ξ_r is then represented in difference form, for ξ_r described as spline coefficients. The splines are selected to automatically satisfy the boundary conditions on ξ_r. The eigenvalue problem is then solved directly, by constructing the characteristic determinant and evaluating its root, ω^2. This technique has been implemented for NRL on the JAYCOR VAX computer system.

The code, EGVPRB, which does this problem is a general eigenvalue solver. It can solve any eigenvalue equation of the form

$$A(r,\lambda)\xi'' + B(r,\lambda)\xi' + C(r,\lambda)\xi(r) = 0$$

where $\xi(r)$ is the eigenfunction, λ is the eigenvalue and prime ($'$) denotes differentiation with respect to r. The codes uses B-spline[5] basis functions, which we denote $\psi_i(r)$. Every function of r is represented by its spline fit,

$$A(r,\lambda) = \Sigma_i a_i(\lambda)\psi_i(r)$$

$$B(r,\lambda) = \Sigma_i b_i(\lambda)\psi_i(r)$$

$$C(r,\lambda) = \Sigma_i c_i(\lambda)\psi_i(r)$$

$$\xi(r) = \Sigma_i \gamma_i \psi_i(r)$$

The differential eigenvalue equation is then

$$\Sigma_{ij} a_i \gamma_j \psi_i \psi_j'' + \Sigma_{ij} b_i \gamma_j \psi_i \psi_j' + \Sigma_{ij} c_i \gamma_j \psi_i \psi_j = 0.$$

Multiplying by ψ_ℓ, for each ℓ value, and averaging over r (denoted by $\langle\rangle$), leads to a matrix equation,

$$\Sigma_j [\Sigma_i a_i \langle \psi_\ell \psi_i \psi_j'' \rangle + \Sigma_i b_i \langle \psi_\ell \psi_i \psi_j' \rangle + \Sigma_i c_i \langle \psi_\ell \psi_i \psi_j \rangle] \gamma_j = 0,$$

or

$$\Sigma_j M_{\ell j} \gamma_j = 0,$$

or

$$\underline{\underline{M}} \cdot \underline{\gamma} = 0.$$

This equation has a solution for $\underline{\gamma}$, provided

$$\det \underline{\underline{M}} = 0,$$

which is solved for the eigenvalue, $\lambda \equiv \omega^2$, using a root-finder.

The code can also be used in a mode which displays the behavior of $A(r,\lambda)$, $B(r,\lambda)$, and $C(r,\lambda)$ vs. r as λ is varied, and which shows det $\underline{\underline{M}}$ vs. λ. The code is described in some detail in Appendix B. A simple test calulation is described here to illustrate the use of the code. An equilibrium consisting of a uniform density plasma cylinder ($n = 10^{18}$ cm^{-3}), of radius, $r_p = 1$ cm, is assumed to carry a uniformly-distributed current, I = 3MA. The magnetic field then rises linearly within the cylinder, achieving a peak value, $B_\theta(r_p) = 60T$ (or 600 KG), at the plasma edge. The plasma is imagined to have a uniform temperature, T = 1KeV, implying a pressure, $p = nK_BT = 1.6 \times 10^8$ nt/m^2.

For a kink mode with m = 1 and $Kr_b = 3$, Figure 3-2

displays the value of the determinant for various trial eigenvalues, where the eigenvalues have been normalized on the plot so that $0.194 \leq (\omega r_b/V_A)^2 \leq 0$. Here V_A is the Alfven speed at the plasma edge. The horizontal line is $\det |\underline{M}| = 0$, and the intersection of the two lines is the root. Figure 3-3 through 3-5 shows the behavior of $A(\omega^2, r)$, $B(\omega^2, r)$, $C(\omega^2, r)$ as ω^2 is varied, in ten equal steps, over the range $-0.194 \leq (\omega r_b/V_A)^2 \leq -0.155$, which includes the root, or to "manually" locate the root.

Having found an approximate root, the root finder in EFVPRB can be utilized to refine the calculation. In this case, the root finder obtained a root at $(\omega r_b/V_A)^2 = -0.183$, in good agreement with Figure 3-2.

REFERENCES

1. E. Hameiri, "The Stability of a Particular MHD Equilibrium with Flow", Courant Institute of Mathematical Sciences, Report C00-3077-123 (June 1976).
2. A. Kadish, Phys. Fluids 23, 1920 (1980).
3. G. Bateman, MHD Instabilities (The MIT Press, Cambridge, MA, c. 1978), Chapters 5 and 6.
4. W. Miner, private communication.
5. L. Schumaker, Spline Functions: Basic Theory (John Wiley & Sons, New York, c 1981) Chapter 4.

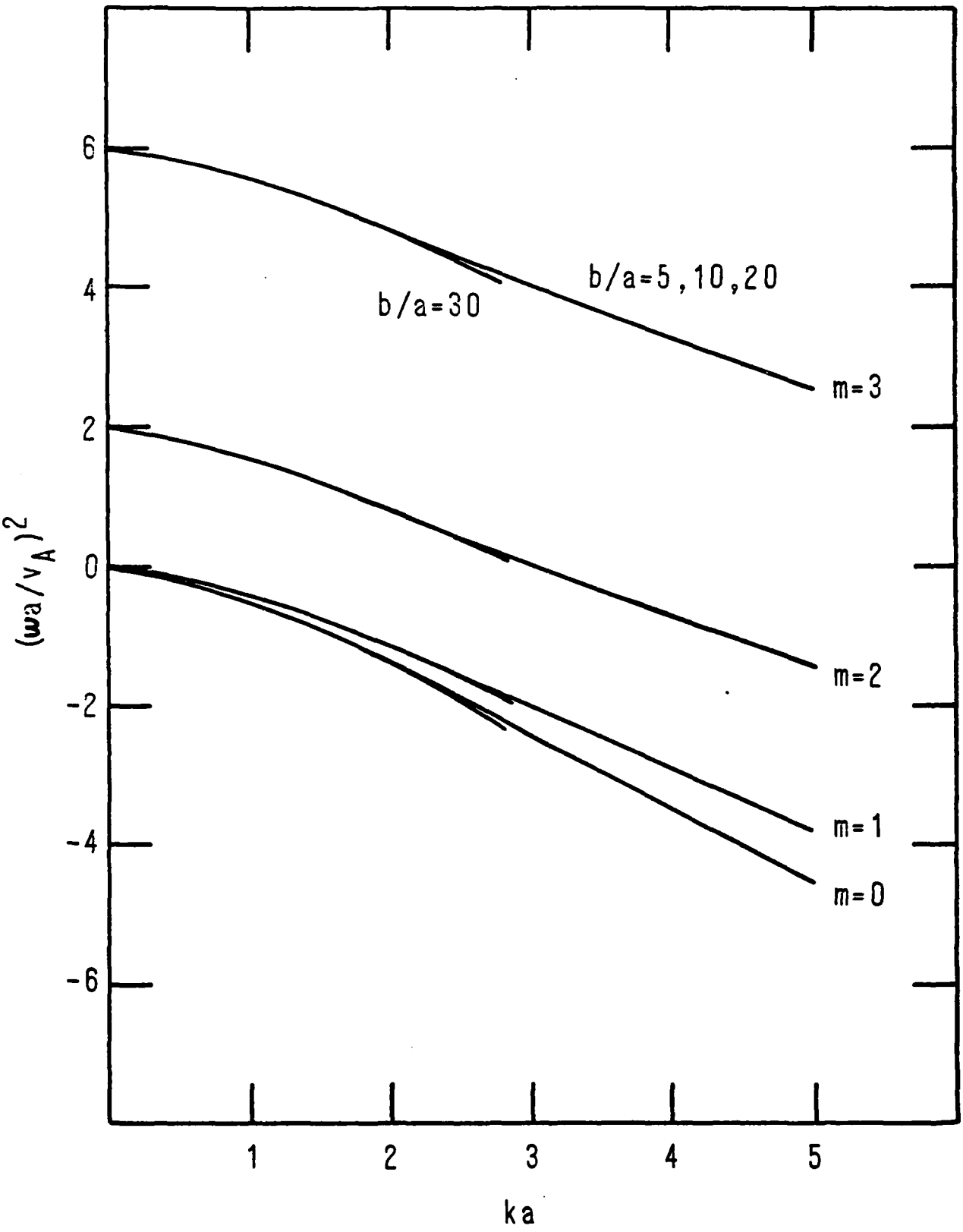


Figure 3-1

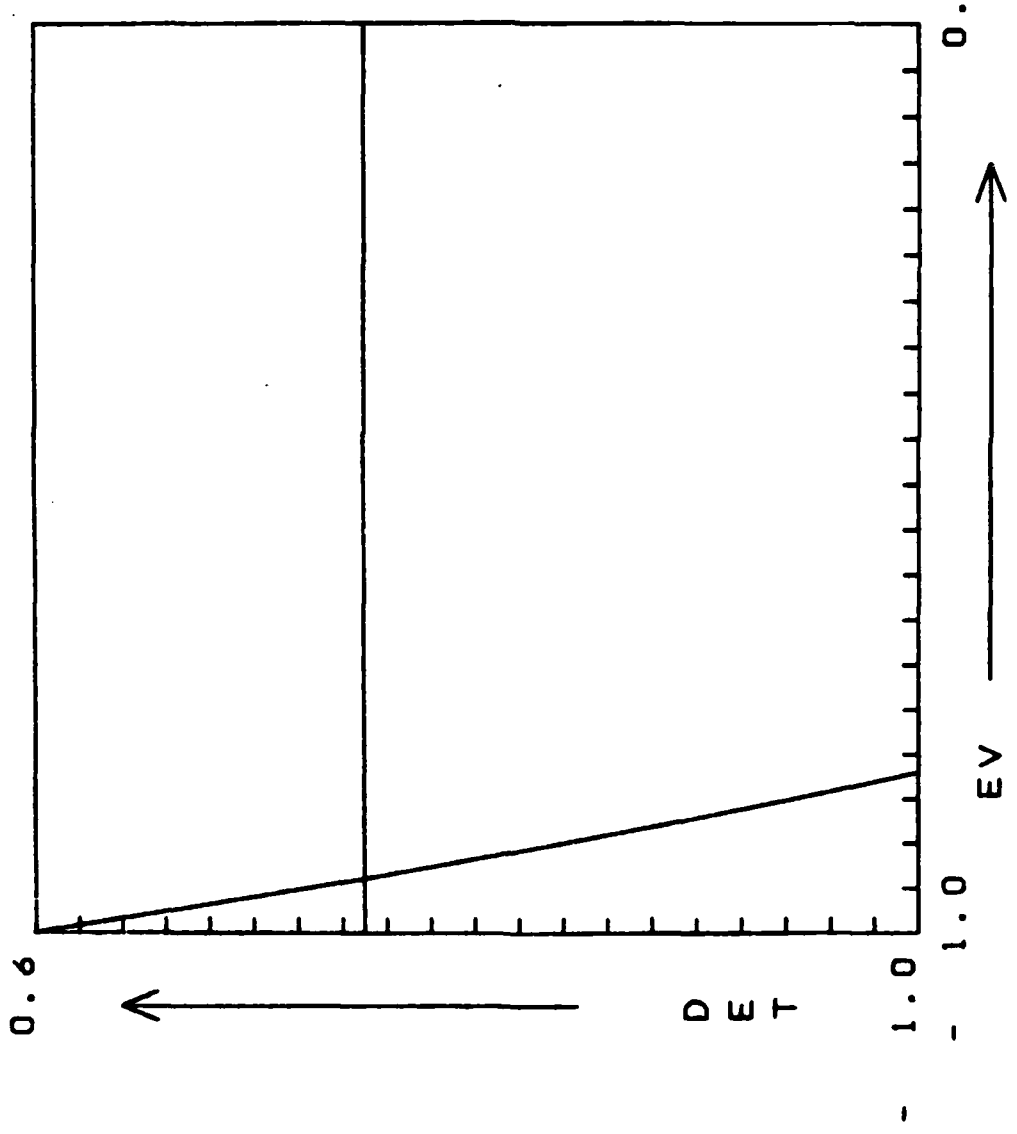


Figure 3-2: det $|\underline{\underline{M}}|$ vs. $(\omega r_b/V_A)^2$; $-0.194 \leq (\omega r_b/V_A)^2 \leq 0$.
Kink Mode: $m=1$, $kr_b=3$.

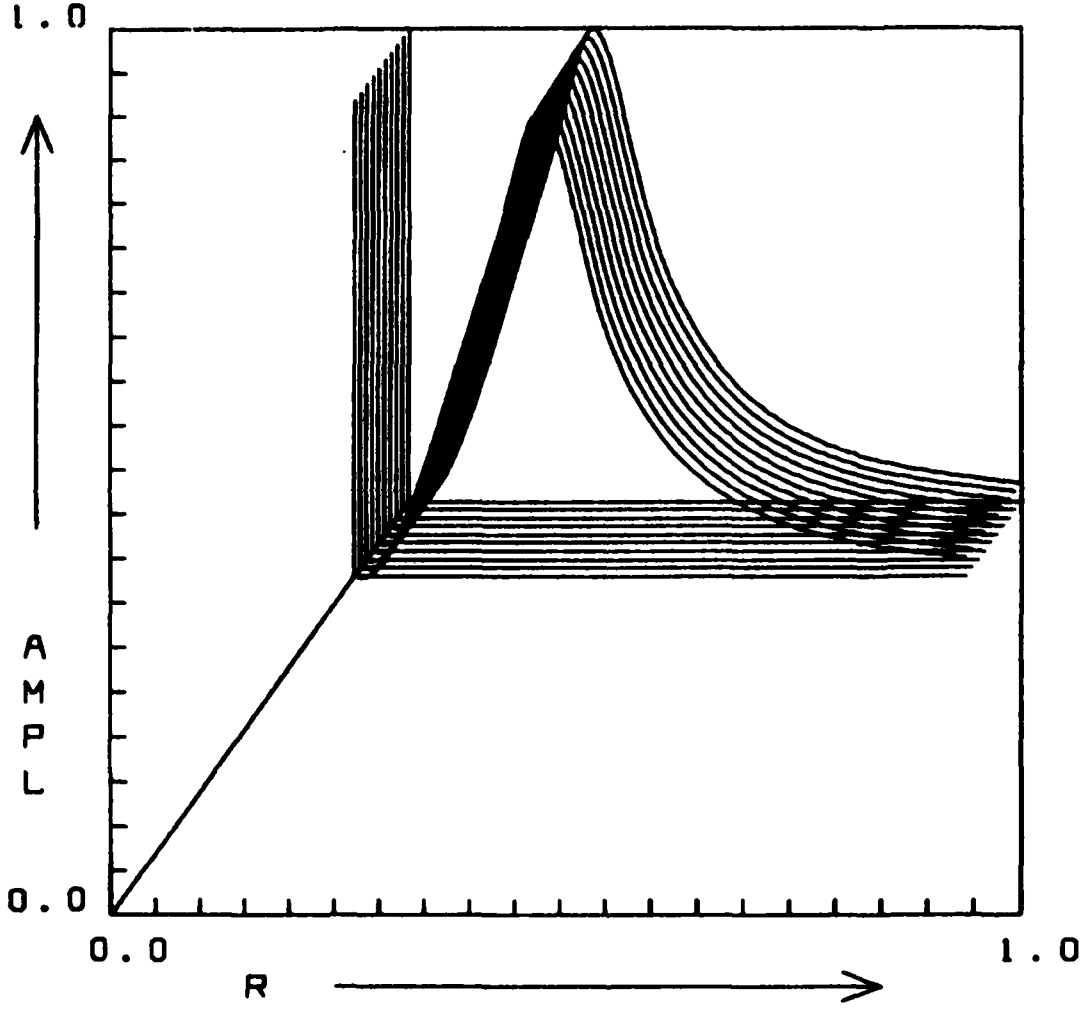


Figure 3-3: Coefficient $A(r,\omega^2)$ vs. r/a for 10 values of ω^2; $-0.194 \leq (\omega r_b/V_A)^2 \leq -0.155$.
Kink Mode: m=1, $Kr_b=3$.

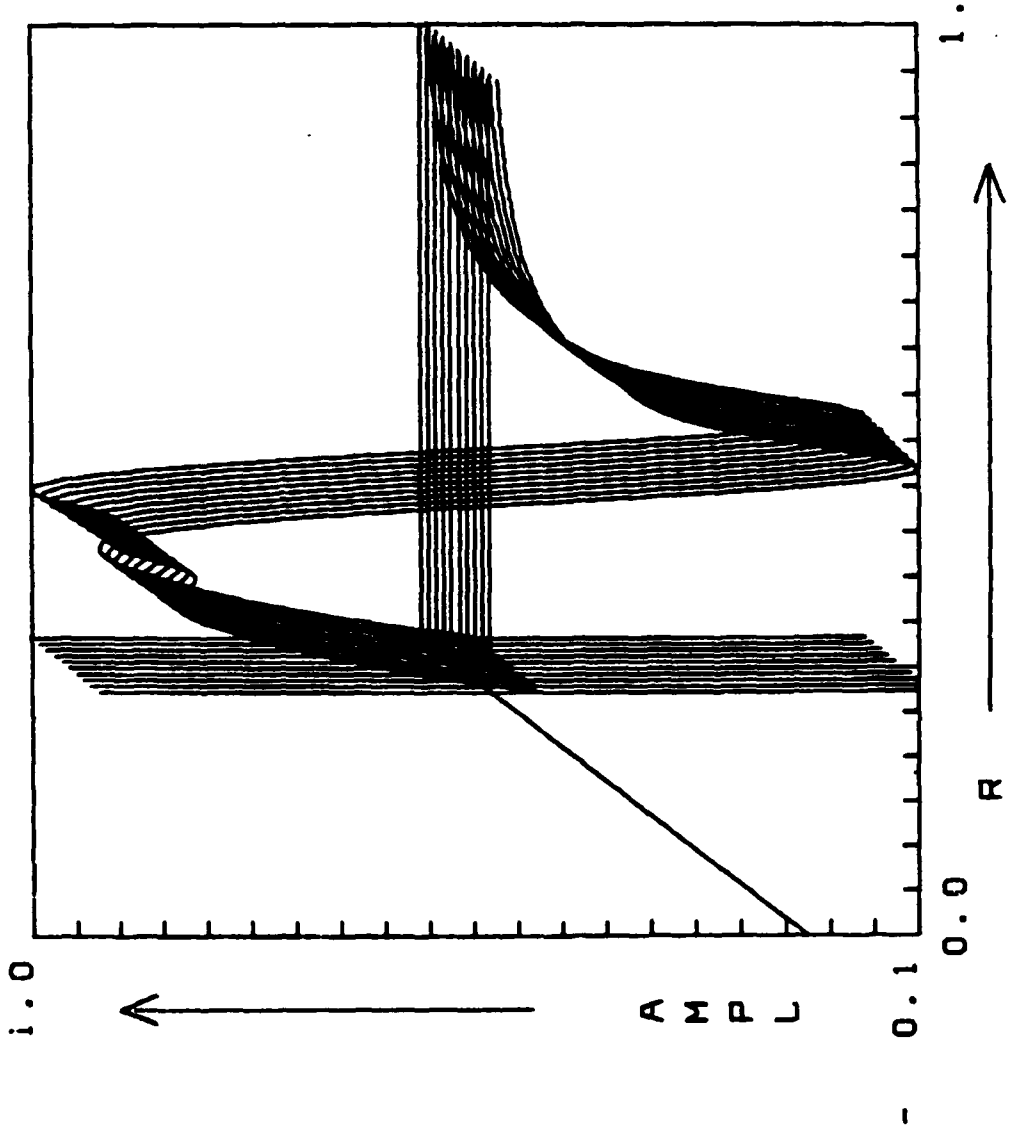


Figure 3-4: Coefficient $B(r,\omega_2^2)$ vs. r/a for 10 values of ω^2; $-0.194 < (\omega r_b/V_A)^2 < -0.155$. Kink Mode: $m=1$, $kr_b=3$.

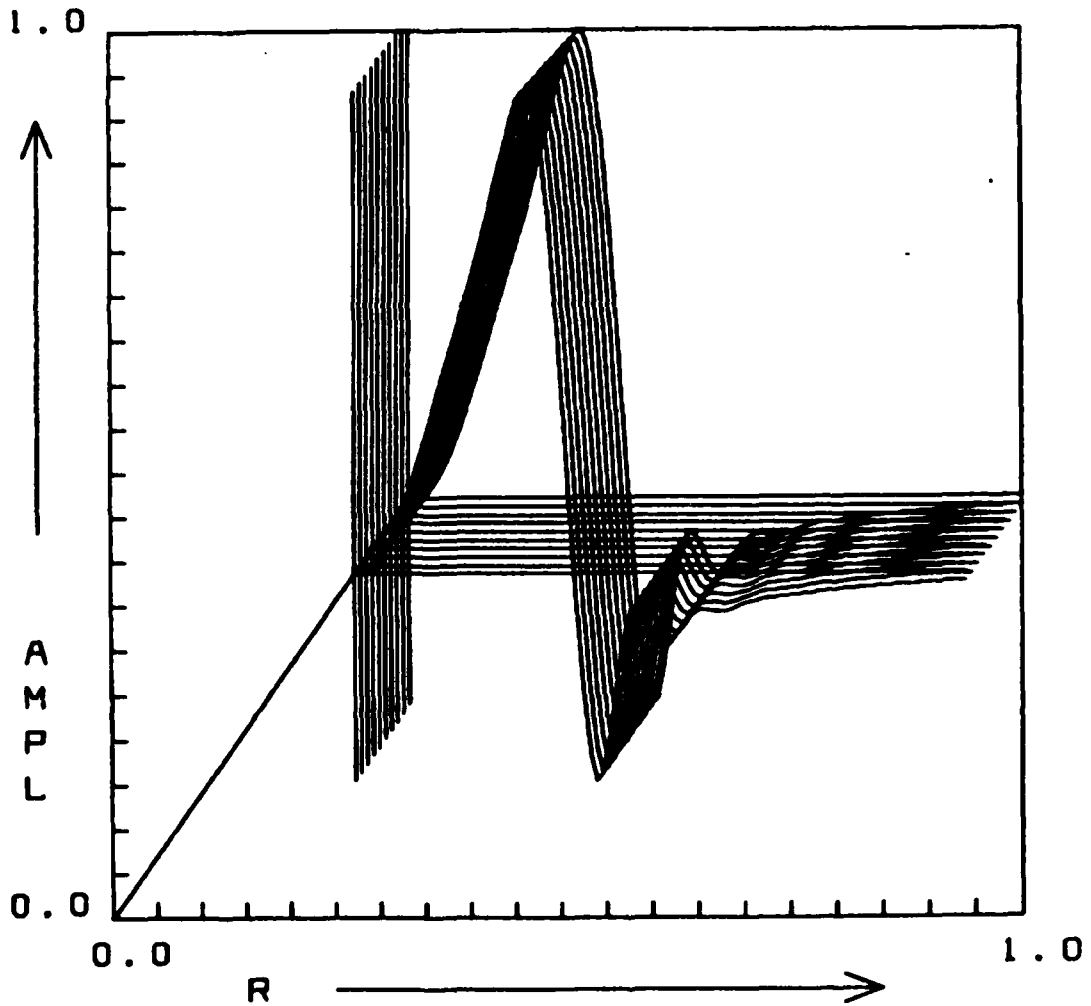


Figure 3-5: Coefficient $C(r,\omega^2)$ vs. r/a for 10 values of ω^2; $-0.194 \leq (\omega r_b/V_A)^2 \leq -0.155$.
Kink Mode: m=1, kr_b=3.

SECTION 4

MAGNETIC INSULATION

A central issue in the scaling of pulsed-power drivers to higher power is whether the vacuum power feed to the diode can withstand the higher electrical stress without loss of magnetic insulation. Magnetic insulation[1-5] refers to the ability of an applied magnetic field to turn emitted electrons back onto the emitting surface, thereby preventing electrical breakdown. This concept is now widely-used in the design of high -power vacuum sections of pulsed-power generators, vacuum transmission lines and ion diodes.[6] The required magnetic field can be applied by external coils, or it can be the self-field due to the current flowing across the anode-cathode gap at the center of the diode.

Maxwell Laboratories, Inc. has conducted a series of tests to study the scaling of the loss current with gap width and electric and magnetic field strength. The apparatus used in the experiments is shown schematically on Figure 4-1. It consists of a disc feed with a variable gap and a 5 nH short-circuit post with a radius of 5 cm. The anode surface is instrumented with a series of Faraday cup collectors, shown schematically on Figure 4-2, located on a radial spoke at various radii. The apertures of the Faraday cups are covered with .003 inch thick aluminum foil to shield the collector from stray plasma, thereby reducing the noise level.

The experiment consists of driving current radially into one disc feed across the short-circuit post, with return current flowing out on the second disc feed. An inductive voltage, LdI/dt, develops across the gap. The current flowing in the post sets up an azimuthal magnetic field which provides magnetic insulation. The object of the experiment is to measure the electron loss current, ie. the current carried by free electrons which cross the magnetically-insulated gap. The Faraday cups provide a measure of this loss.

The four Faraday cups used in the experiment are located at the following locations

FC#1	64cm
FC#2	52cm
FC#3	38cm
FC#4	25cm

each having a collecting area of 1.8 cm^2. The baffle holding the foil, shown on Figure 4-2, limits the incident angle for a trajectory to reach the collector. Ignoring scattering in the foil, only particles with incident angle, θ, relative to the normal to the foil of less than approximately 75° will reach the collector.

Data has been recorded for twenty-one different configurations, characterized by various gap widths and driving currents. In this report an analysis of one of these configurations is presented. The analysis has been carried out with the MASK code, a two-dimensional, fully-relativistic, electromagnetic, particle-in-cell (PIC) plasma simulation code developed by A. Drobot of SAI, in collaboration with NRL, MIT and Lawrence Livermore National Laboratory.

The data for this shot, Shot Number 1105, was graciously supplied by John Shannon of Maxwell Laboratories, and is summarized in Table 4-1. The time history data for this shot is shown on Figures 4-3 through 4-5. The drive current, I(t), is shown on Figure 4-3. Figure 4-4 is dI/dt, while Figure 4-5 shows current, voltage and power.

Figure 4-6 shows the Faraday cup and PIN diode waveforms for this shot. The PIN diodes, located adjacent to the Faraday cups, provide local x-ray data, from which information about the energy spectrum of the loss electrons can be inferred. In this case, the PIN diodes indicated the presence of electrons with energy in excess of 100 kev, but the fraction of electrons above this energy is not known from this diagnostic.

The voltage waveforms for the Faraday cups are the voltage developed by the Faraday cup current as it passes through a 50 Ω termination, with a ten-fold attenuator.

The voltage, V_{sc}, indicated on the scope can be translated to Faraday cup current, I_{fc}, by

$$I_{fc}(\text{amps}) = \frac{10V_{sc}(\text{volts})}{50\Omega} \quad .$$

The configuration shown in Figure 4-1, using a 5 cm gap, has been set-up with the MASK code, and gridded on a 64x16 r-z mesh. The experimental current waveform, Figure 4-3, is used to drive the simulation. The first test of the numerical model is a "cold test", which refers to a run in which no free electrons are allowed to be emitted. This type of run tests the circuit model. Without current smoothing, the calculated induced voltage shows alot of hash. By smoothing the current waveform in Figure 4-3 to relax sudden changes in the current, the calculated voltage is found to be in excellent agreement with the experimental waveform.

The cold test run also provides a wealth of information about the field structure in the device without particles. Some of this data is shown on Figures 4-7 through 4-10, and may be used in conjunction with later figures to examine the effect of including the emitted electrons. On these figures, the coordinates are numbered as $X_1 = Z$, $X_2 = r$, $X_3 = \theta$, so that $E_1 = E_z$, $E_2 = E_r$ and $B_3 = B_\theta$.

Figure 4-7 shows the electric and magnetic field spatial profiles, as contours on the r-z grid, and as a vector plot for the electric field lines (the magnetic field lines are purely azimuthal). These figures represent the system at time t = 108 ns, but they are quite insensitive to time. After the current maximum, dI/dt switches sign, and at later times the electric field vector plot shows arrows pointing opposite to those on Figure 4-7.

The electric and magnetic field components E_z and B_θ are plotted against time for various radii on Figure 4-8 and 4-9. These plots show the shape of the driving current waveform (since $B_\theta \propto I$) and the shape of the induced voltage waveform (since $E_z \propto V \propto dI/dt$).

Figure 4-10 shows the total field energy against time. It also displays the breakdown of field energy associated with each field component. The curves have the expected shape for an inductively driven gap.

The next step is to turn on electron emission on the cathode. To compare with Maxwell's data, the anode surface includes absorber regions which collect charge as a Faraday cup. The absorbers in the code are rings located on the anode surface at the same radii as the Faraday cups in the experiment. Each absorber ring is four cells (or 4.12 cm) thick with radius R, and therefore presents a collecting area of $2\pi R \times 4.12$ cm^2, which must be renormalized

to the 1.8 cm^2 collecting area of the Faraday cups used in the experiments. The code does not include the .003 inch aluminium foil or the baffle on which the foil is stretched (see Figure 4-2). All charges which hit the absorber regions are collected and counted, whereas the experiment only counted those electrons with sufficient energy to penetrate the foil and with incident angles less than approximately 75° to the normal.

The results of the simulation are summarized on Figures 4-11 through 4-24. Figure 4-11 shows the field spatial behavior at two separated instants of time, t = 160 ns and t = 220 ns. The earlier time shows a field structure, which is similar to the cold-test field (cf. Figure 4-7), except for the effect of particles near the short-circuit post. The late-time field structure shows the effect of particle emission. Figure 4-12 shows the total field energy and the field energy associated with each field component plotted against time. These plots represent the time dependence of the volume-averaged fields. The corresponding plots in the absence of emitted electrons are shown on Figure 4-10. The field energy is dominated by the energy stored in the B_θ field.

The particle density on the grid is displayed on Figure 4-13 at t = 160 ns, 220 ns, and 340 ns, which shows the development of the electron loss current in the gap.

The total, volume-integrated charge in the system plotted against time is also shown on Figure 4-13. The maximum total charge in the gap occurs at t = 160 ns, but is highly localized near the cathode surface.

The particle phase space projections are given on Figures 4-14 through 4-18, for time t = 160 ns, 240 ns and 340 ns. Figures 4-14 and 4-15 illustrate the behavior of the axial momentum P_z vs. z (or X_1) and P_z vs. r (or X_2), and show the development of the electron loss current from magnetically-insulated emission on the cathode to the formation of an electron layer throughout the gap, due partially to electron emission from the short-circuit post. Figures 4-16 and 4-17 show the same information for the radial momentum, P_r, while Figure 4-18 shows P_r vs P_z. The electron loss begins on the cathode at large r, where the magnetic insulation is weakest, and gradually progresses toward the short-circuit post, since the current crossing the gap at large r reduces the magnetic field at small r.

The $\underline{E}\cdot\underline{J}$ instantaneous power is plotted against time on Figure 4-19, which shows the total power, as well as the contributions due to the radial and axial components. The axial component, $E_z J_z$, is the dominant one, as expected for current loss across the gap.

Electron emission in the code is allowed from both the cathode surface (or right-hand boundary) and the surface

of the short-circuit rod (or lower boundary). Figure 4-20 shows the emitted current from these two surfaces plotted against time.

The electrons may be collected on all surfaces. Figures 4-21 shows the collected current vs. time on the anode (left), cathode (right), top (upper) and short-circuit-rod (lower) surfaces.

The instantaneous current collected by absorber number 1 through 4, which correspond to Faraday cups 1 through 4, is shown on Figure 4-22. The total integrated current, and the current integrated over 1000 time steps (10^{-8} sec) is shown on Figure 4-23 for Faraday cup number 1 and on Figure 4-24 for Faraday cup number 2.

The calculated current on Figure 4-24 has two major discrepancies with the experimental data. First, the waveform for the current averaged over 10 ns bins shows two peaks separated by almost 200 ns in time. The experimental Faraday cup waveform does not show such widely separated peaks. Second, the magnitude of the integrated loss as calculated by the code is approximately 600 times greater than the experimentally-measured loss.

Both of these discrepancies can be attributed to the absence in the code of the .003 inch aluminium foil covering the Faraday cup and the geometrical aperture caused by recessing the Faraday cup charge collector as

shown on Figure 4-2. The aluminium foil will stop a 130 keV electron at normal incidence. A grazing electron with much lower energy will stop in the foil.

An examination of the particle energy spectrum computed by the code indicates that essentially all of the electrons which make up the first spike on Figure 4-24 lie below 100 keV and therefore would not be detected in the experiment. Half of the electrons in the second, later spike on Figure 4-24 also lie below 100 keV. The other half are energetic enough to produce the PIN diode signal observed in the experiment.

The electron orbits as they impinge on the absorbers (Faraday cups) in the code are very steep since the electrons drift radially-inward as they traverse the gap, due to the $E_z \times B_\theta$ drift. Most of these electrons are therefore blocked by the Faraday cup acceptance ($\theta \lesssim 75°$) aperture, or are stopped in the aluminium foil.

Modifications to MASK are currently in progress to quantify these effects. The results to date, however, indicate that the electron losses measured in the Maxwell experiment probably constitute only a small fraction of the actual loss present in the apparatus. The scaling of existing pulse-power devices to significantly higher power will depend on the understanding and control of these power losses in the vacuum sections of the machines.

REFERENCES

1. A. Ron, A. Mondelli and N. Rostoker, IEEE Trans. on Plasma Sci. PS-1, 85 (1973).
2. R.V. Lovelace and E. Ott, Phys. Fluids 17, 1263 (1974).
3. V.S. Voronin and A.N. Lebedev, Sov. Phys.-Tech. Phys. 18, 1627 (1974).
4. J.M. Creedon, J. Appl. Phys. 46, 2946 (1975).
5. J.M. Creedon, J. Appl. Phys. 48, 1070 (1977).
6. Proc. 4th Int'l Top. Conf. on High-Power Electron and Ion-Beam Research and Techology (Palaiseau, France, 1981).

TABLE 4-1

MAXWELL DATA FOR SHOT 1105

Radius (cm)	Gap (cm)	L (nH)	Q (nC/cm^2)	V (MV)	I (MA)	E (MV/m)	B (Tesla)
64.2	5	37.40	1.18	.069	2.24	1.38	0.698
52.3	5	35.35	0.78	.021	2.25	0.42	0.860
37.9	5	32.13	9.89	.523	2.10	10.46	1.108
25.4	5	28.13	54.32	.458	2.10	9.16	1.653

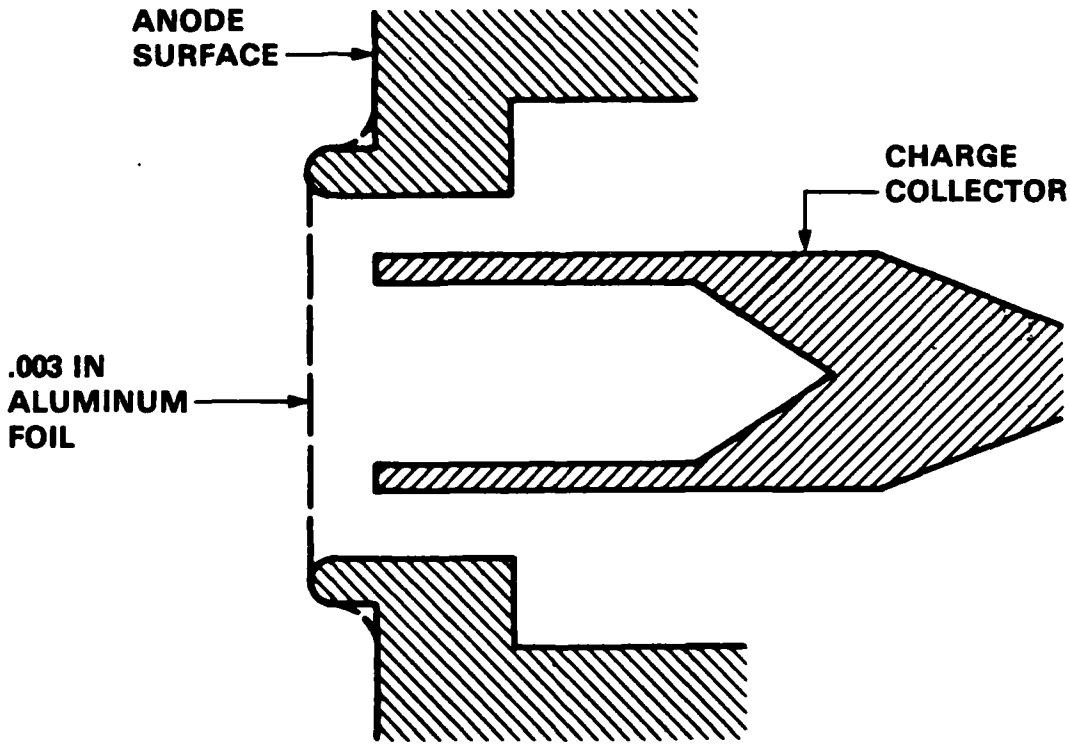


FIGURE 4-2: FARADAY CUP

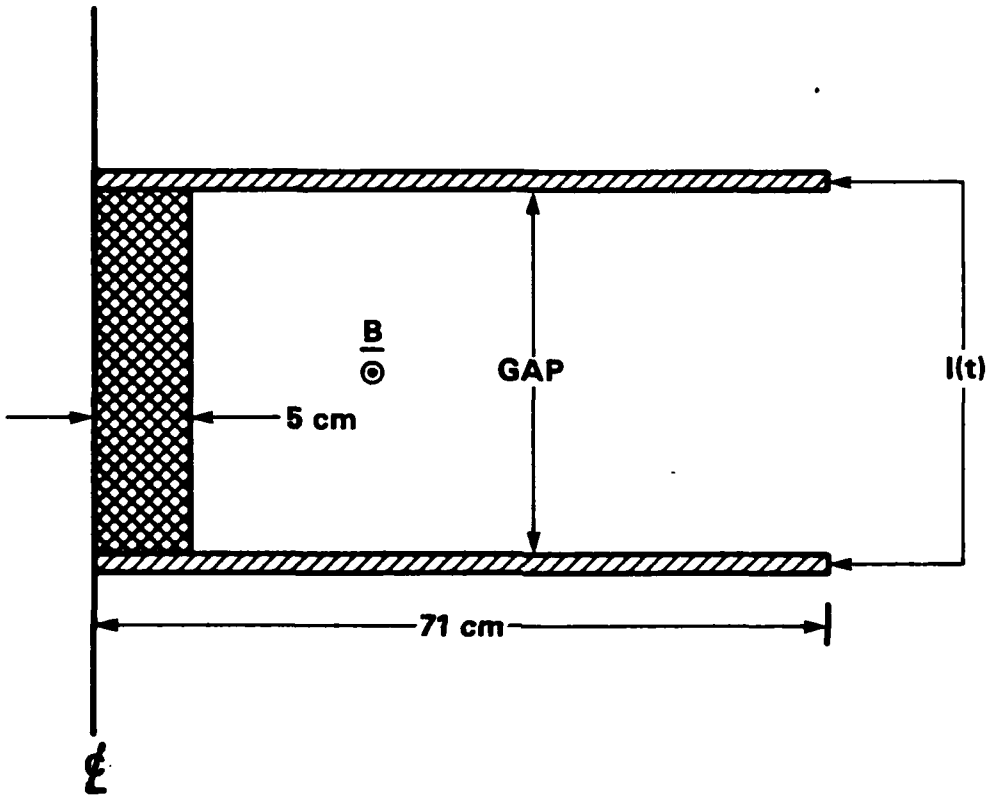


FIGURE 4-1: SCHEMATIC DRAWING OF EXPERIMENT

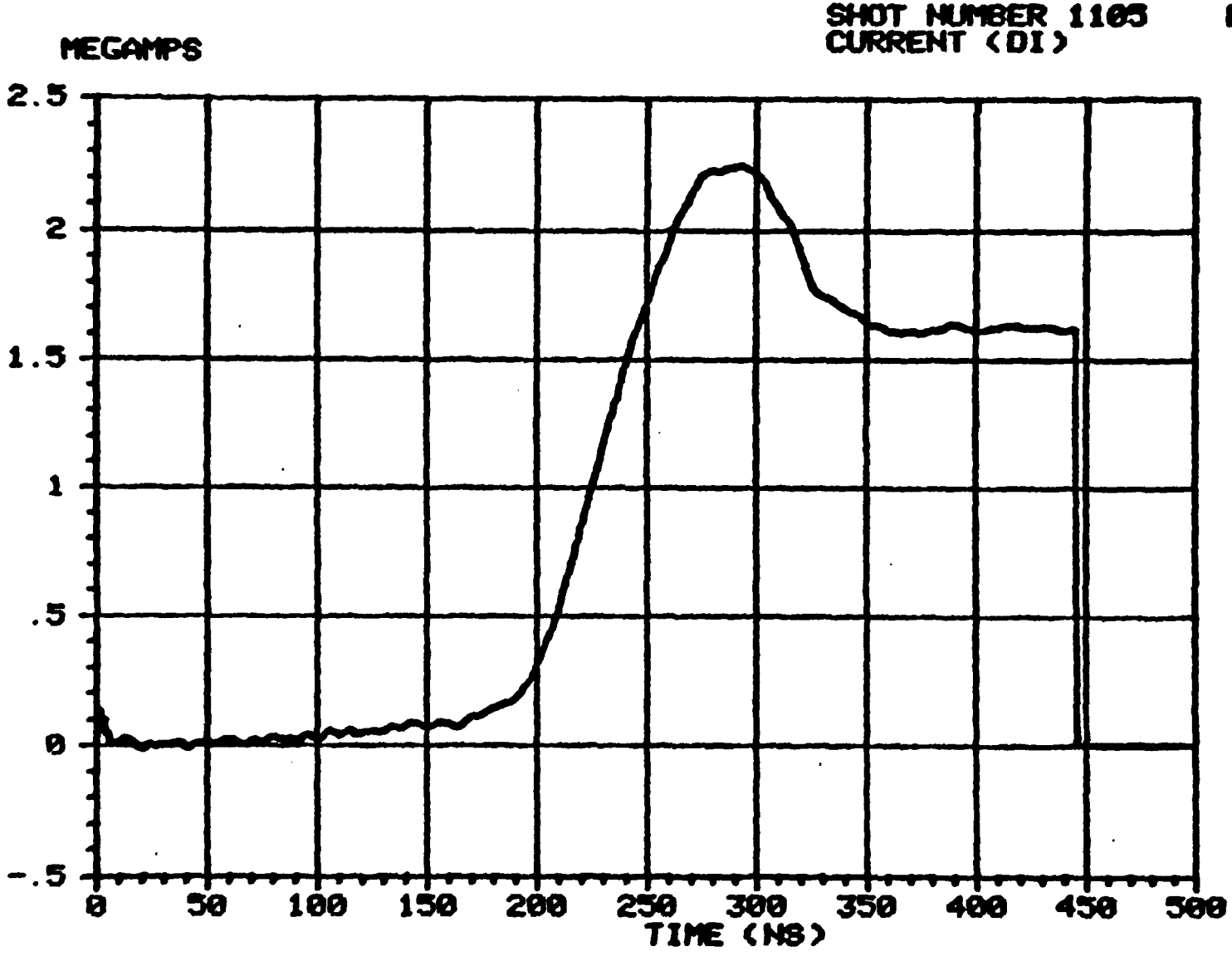


Fig. 4-3: Experimental Current Waveform

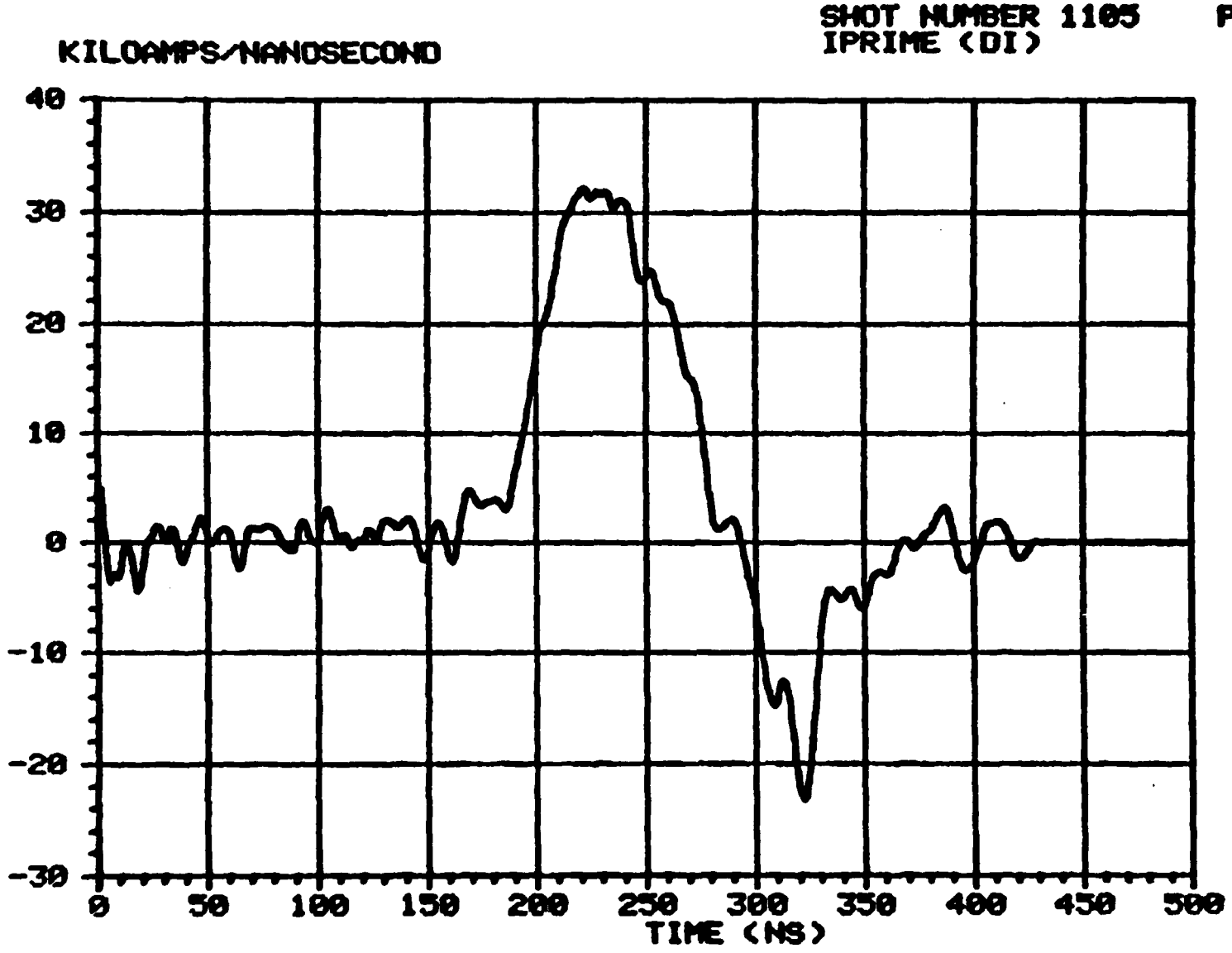


Fig. 4-4: Experimental dI/dt Waveform

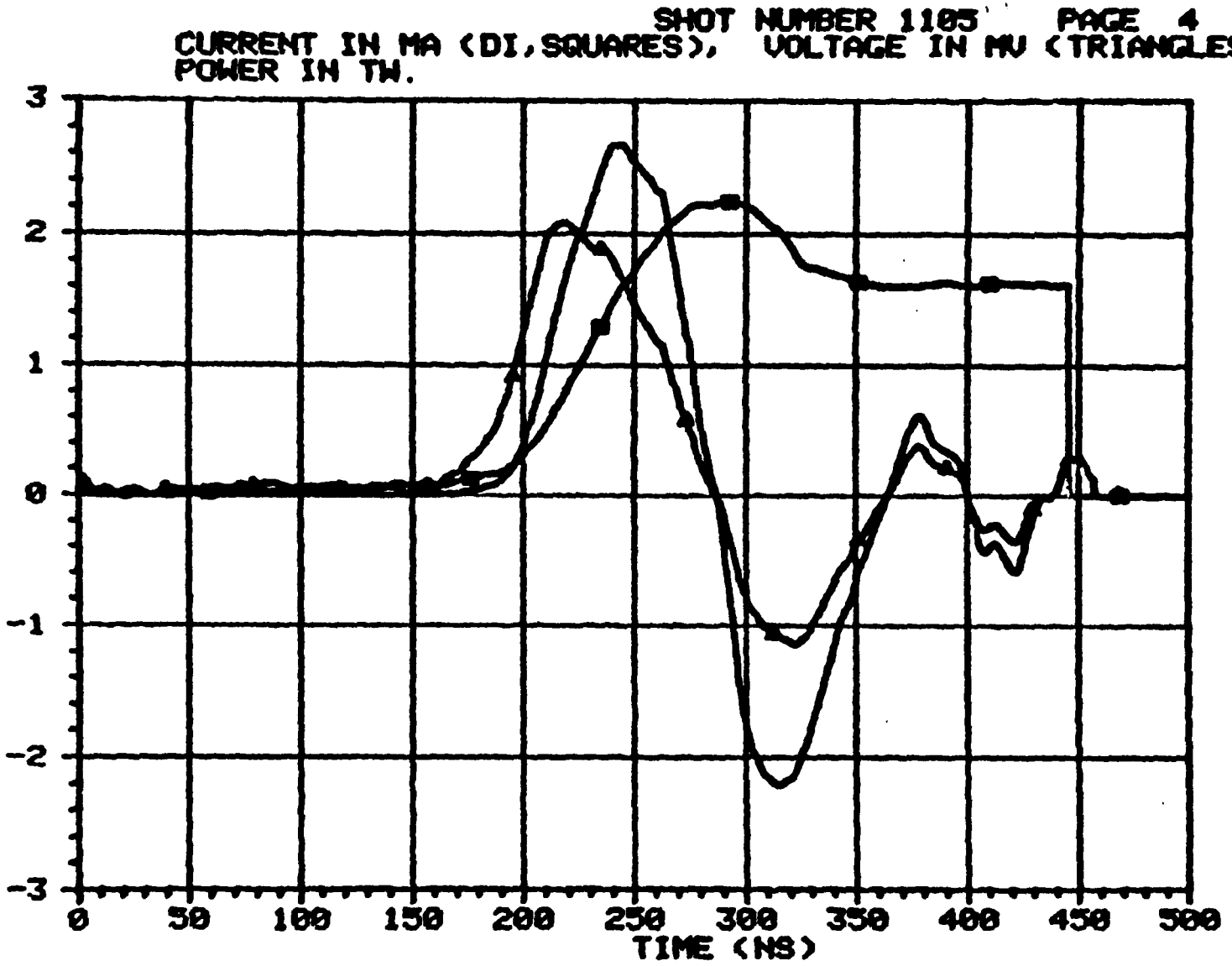
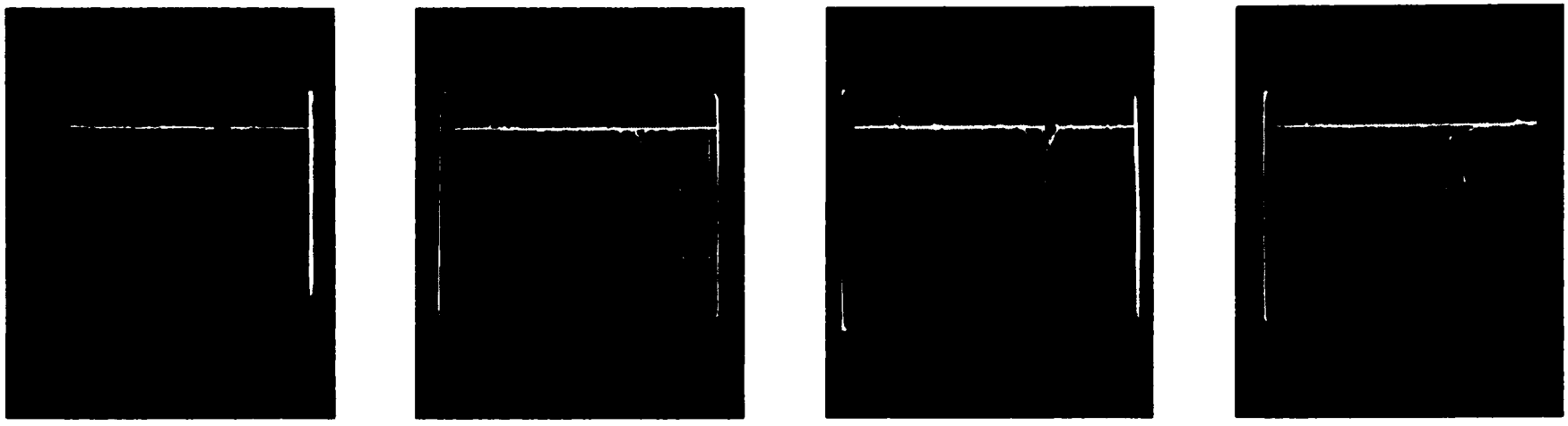


Fig. 4-5: Measured Current (Squares), Voltage (Triangles) and Power (Solid) Waveforms

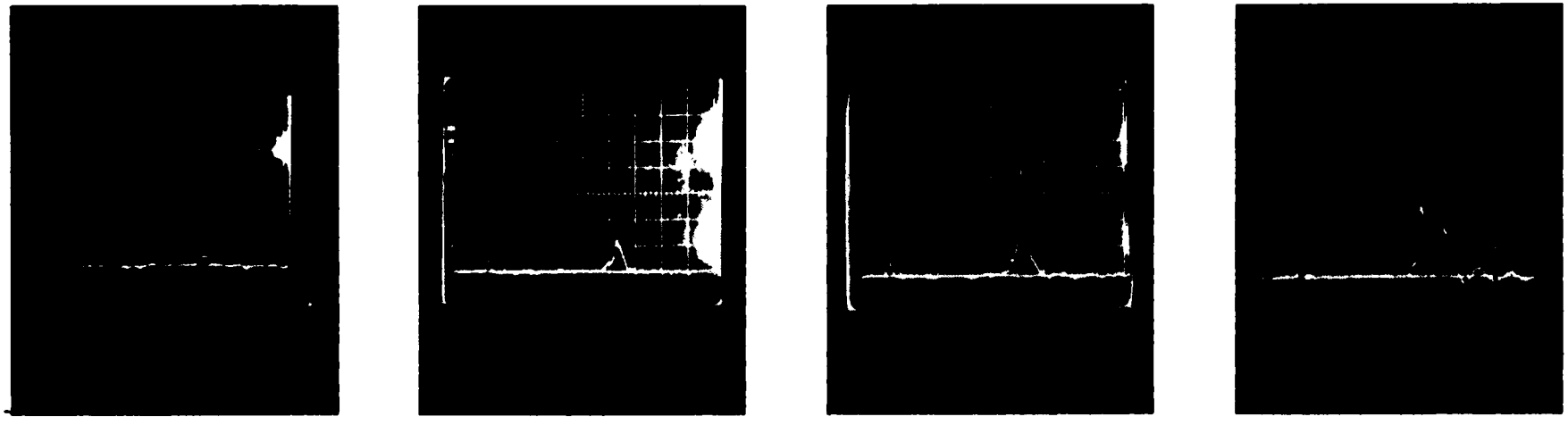
MAXWELL SHOT NO. 1105

(a) FARADAY CUPS



#1 FC1 .5V/DIV #2 FC2 .5V/DIV #3 FC3 1V/DIV #4 FC4 2V/DIV

(b) PIN DIODES



#1 PD1 .5V/DIV #2 PD2 .5V/DIV #3 PD3 1V/DIV #4 PD4 2V/DIV

FIGURE 4-6: FARADAY CUP AND PIN DIODE WAVEFORMS

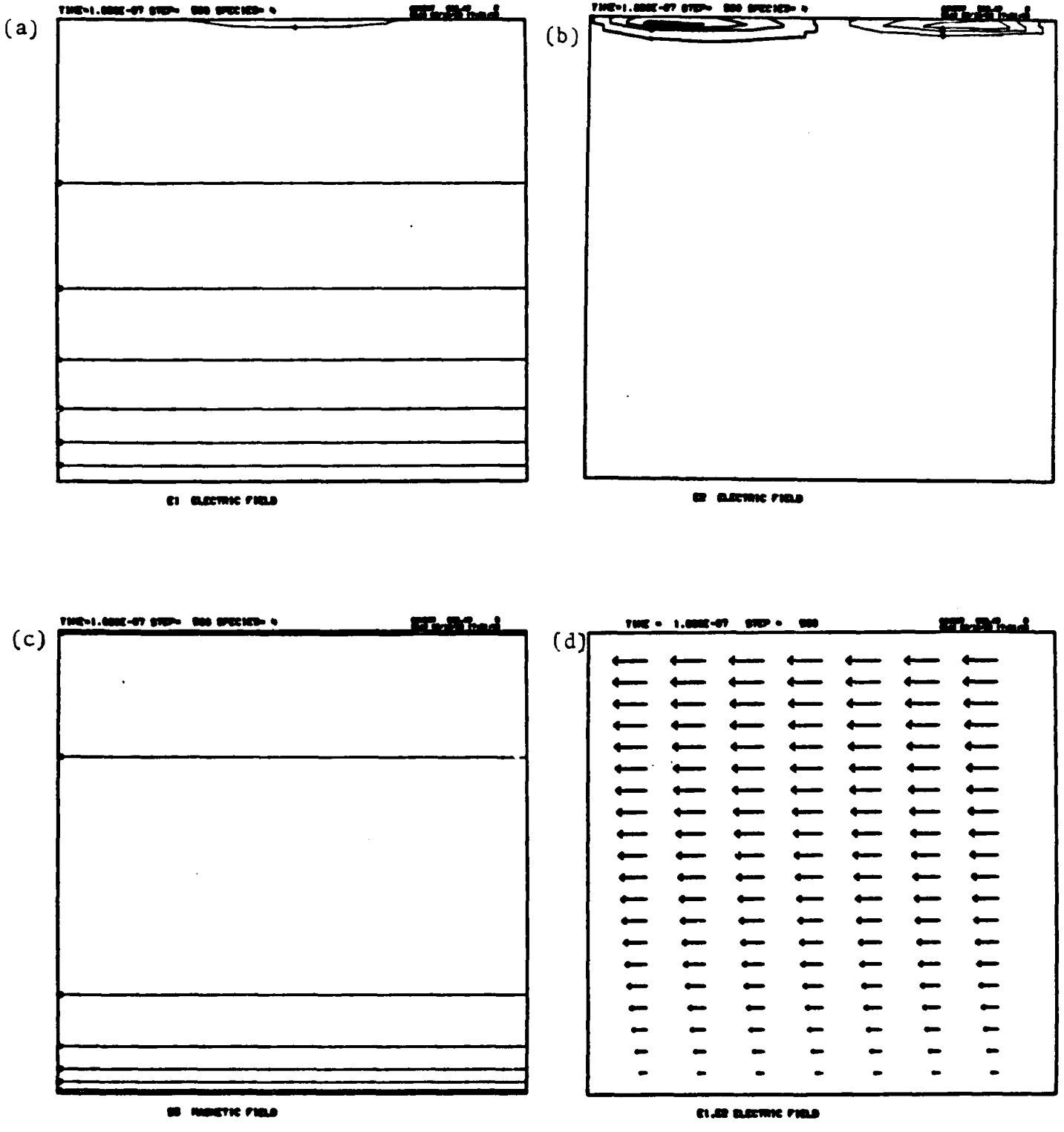


Fig. 4-7: Cold Test Field Structure at t= 108 ns.

(a) E_z Contours; (b) E_r Contours;
(c) B_θ Contours; (d) $\underline{E}$ vector plot.

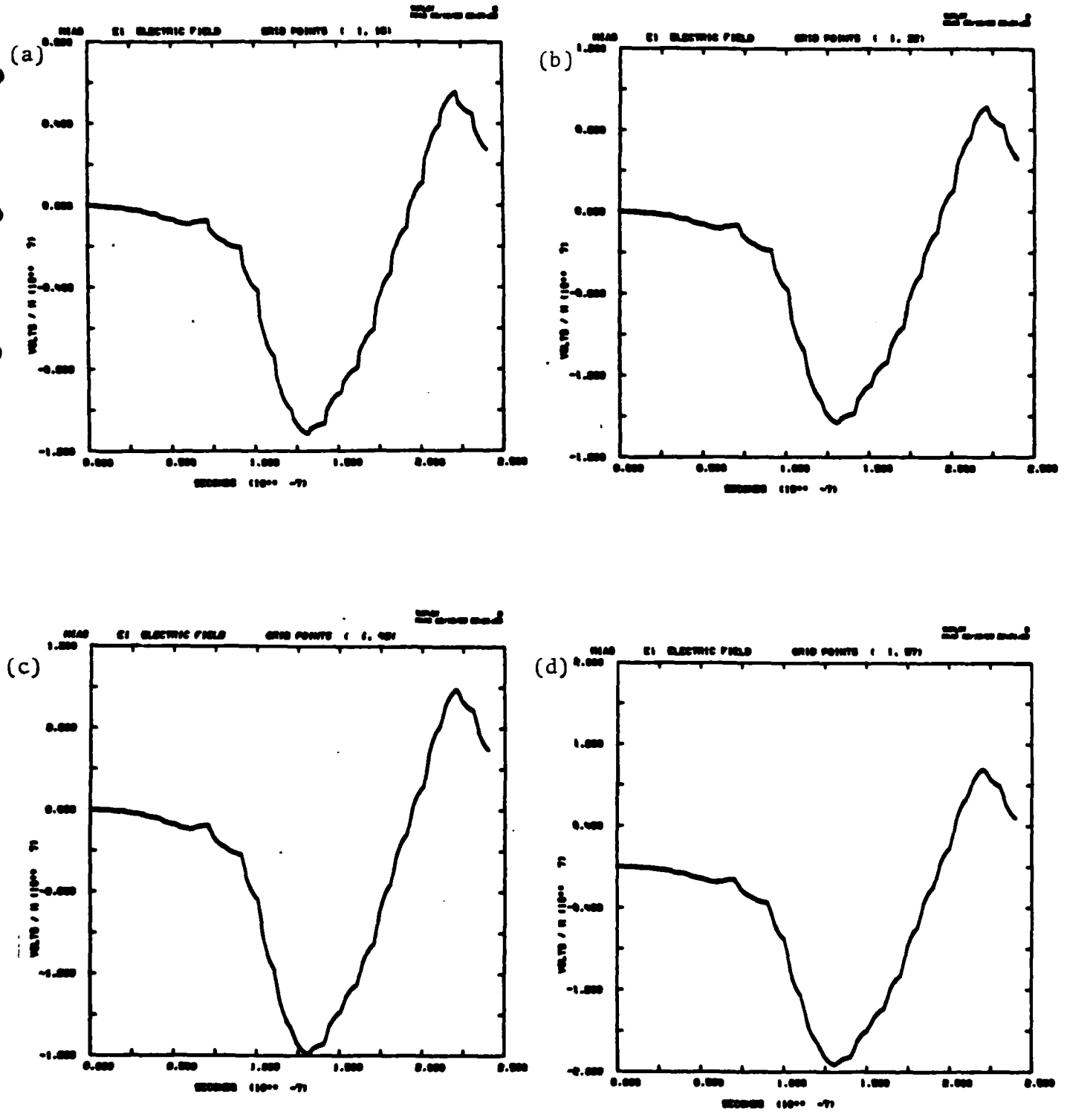


Fig. 4-8: Cold Test E_z vs. time on the Anode Surface.

(a) r= 21.5 cm; (b) r=38.0 cm;
(c) r= 52.4 cm; (d) r=63.7 cm.

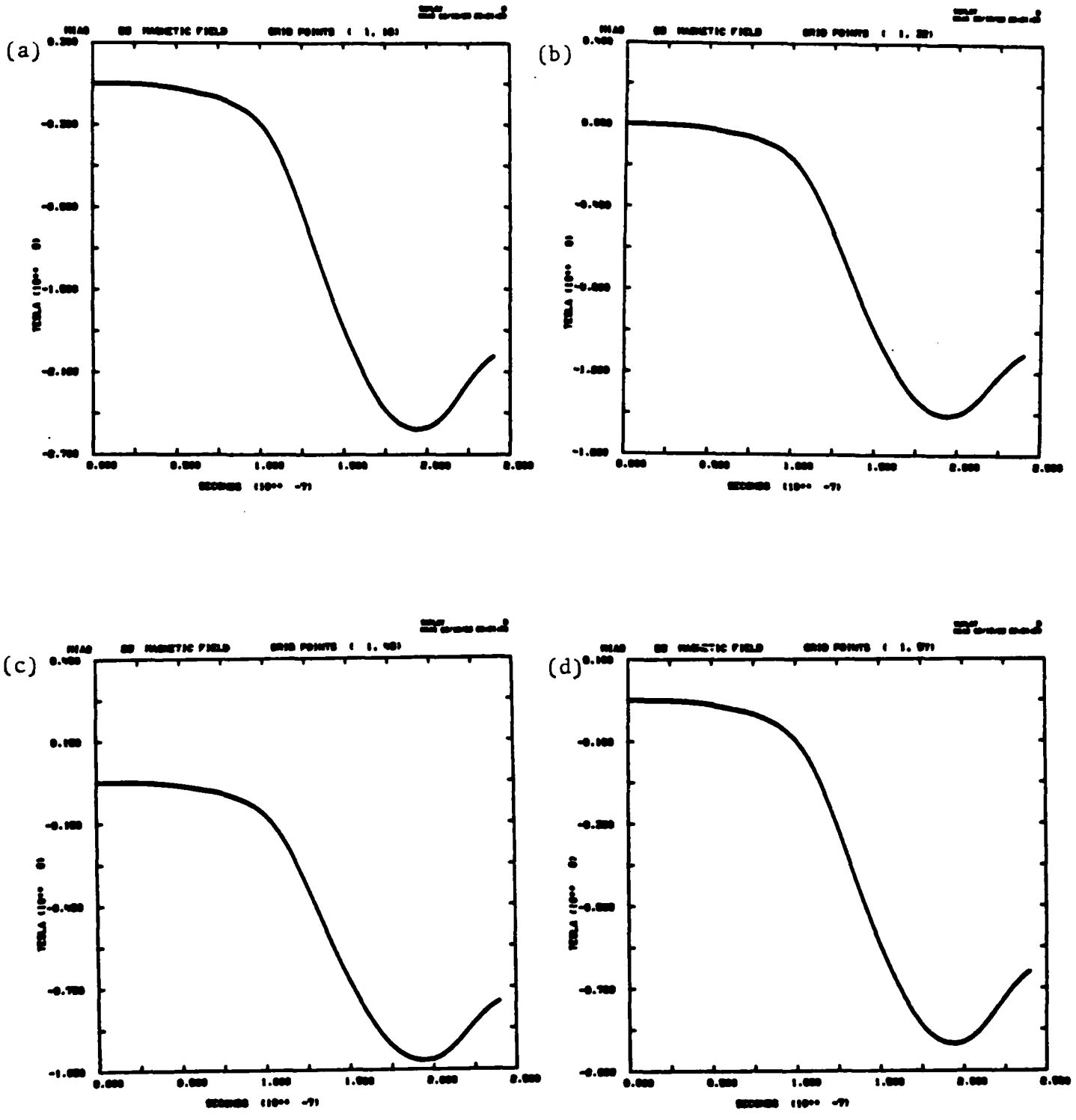


Fig. 4-9: Cold Test B_θ vs. time on the Anode Surface.

(a) r=21.5 cm; (b) r=38.0 cm;
(c) r=52.4 cm; (d) r=63.7 cm.

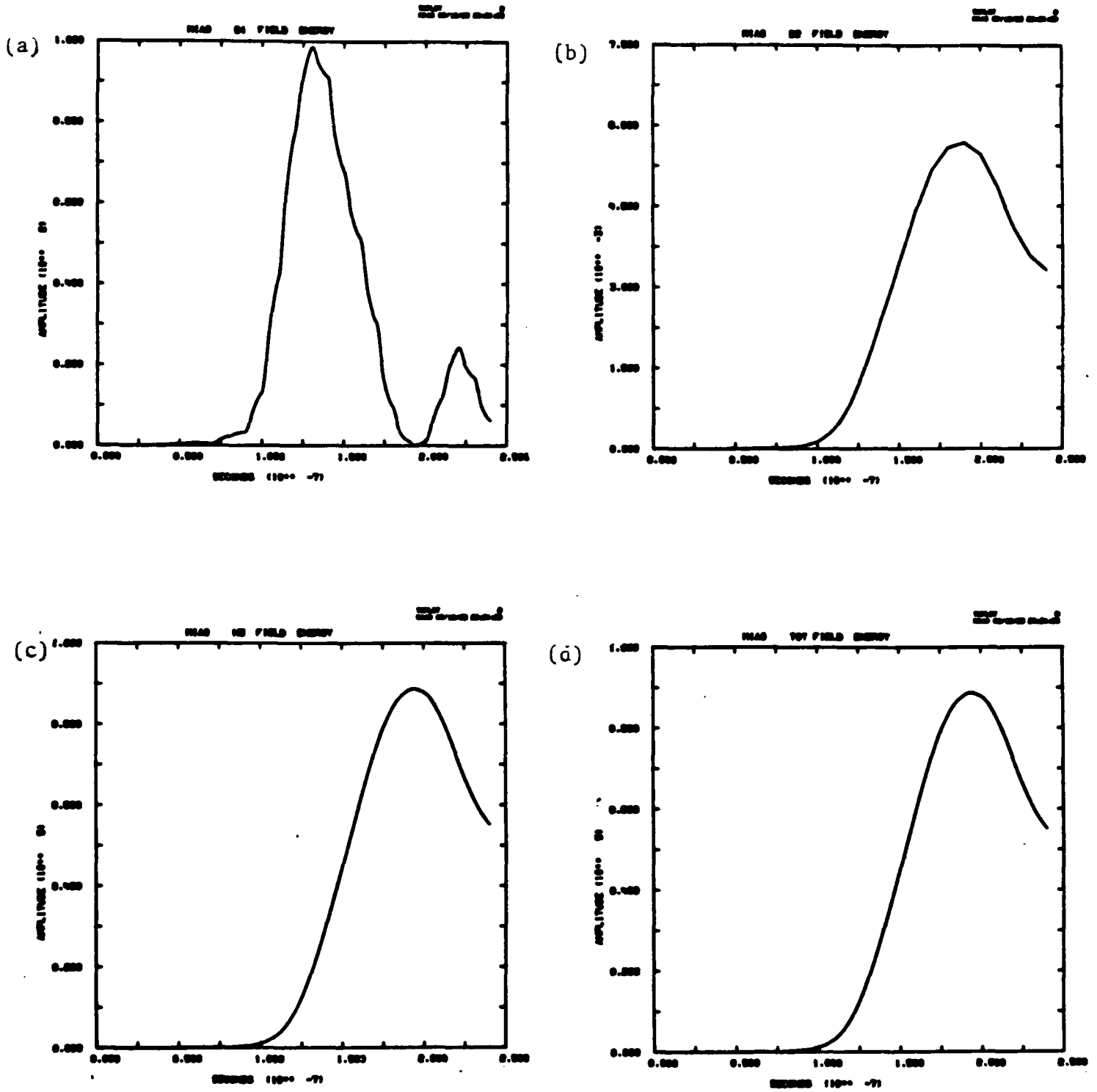


Fig. 4-10: Cold Test Field Energy vs. Time.

(a) D_z Component; (b) D_r Component;
(c) H_θ Component; (d) Total Field Energy.

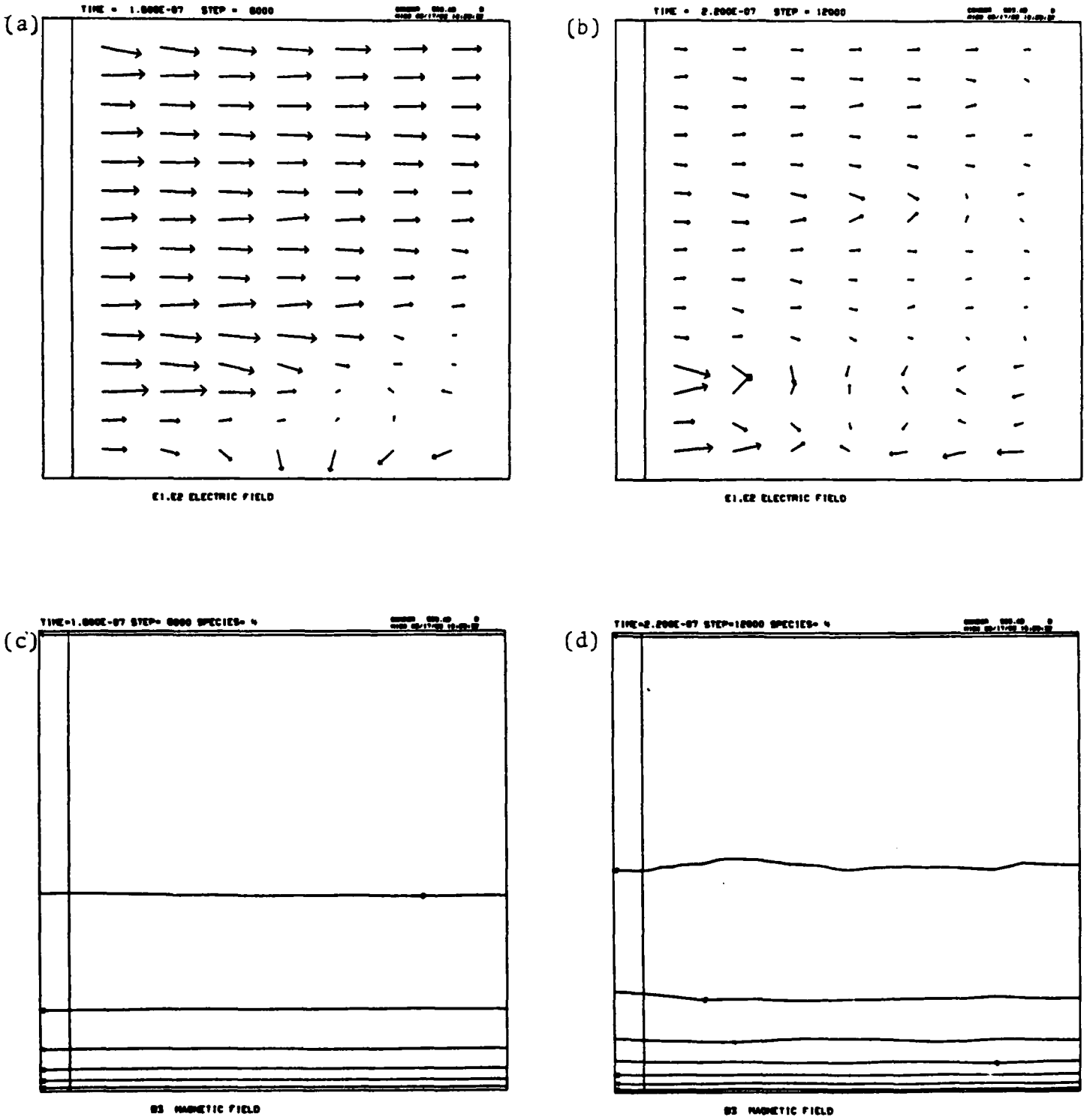


Fig. 4-11: Electric and Magnetic Fields.

(a) $\underline{E}$ vectors at t=160 ns;
(b) $\overline{E}$ vectors at t=220 ns;
(c) $\overline{B}_\theta$ Contours at t=160 ns;
(d) B_θ Contours at t=220 ns.

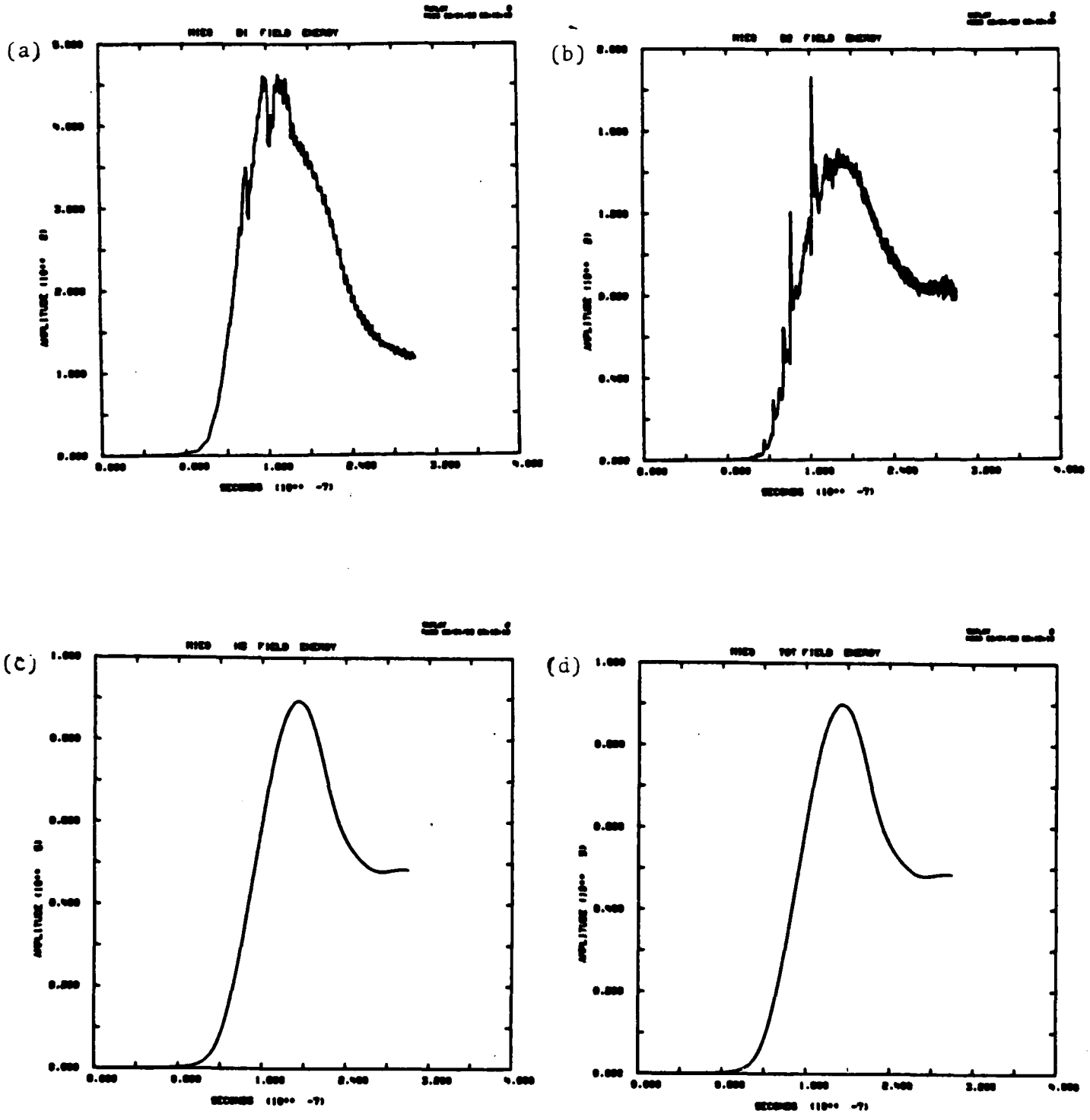


Fig. 4-12: Stored Field Energy vs. time.

(a) D_z Component; (b) D_r Component;
(c) H_θ Component; (d) Total Field Energy.

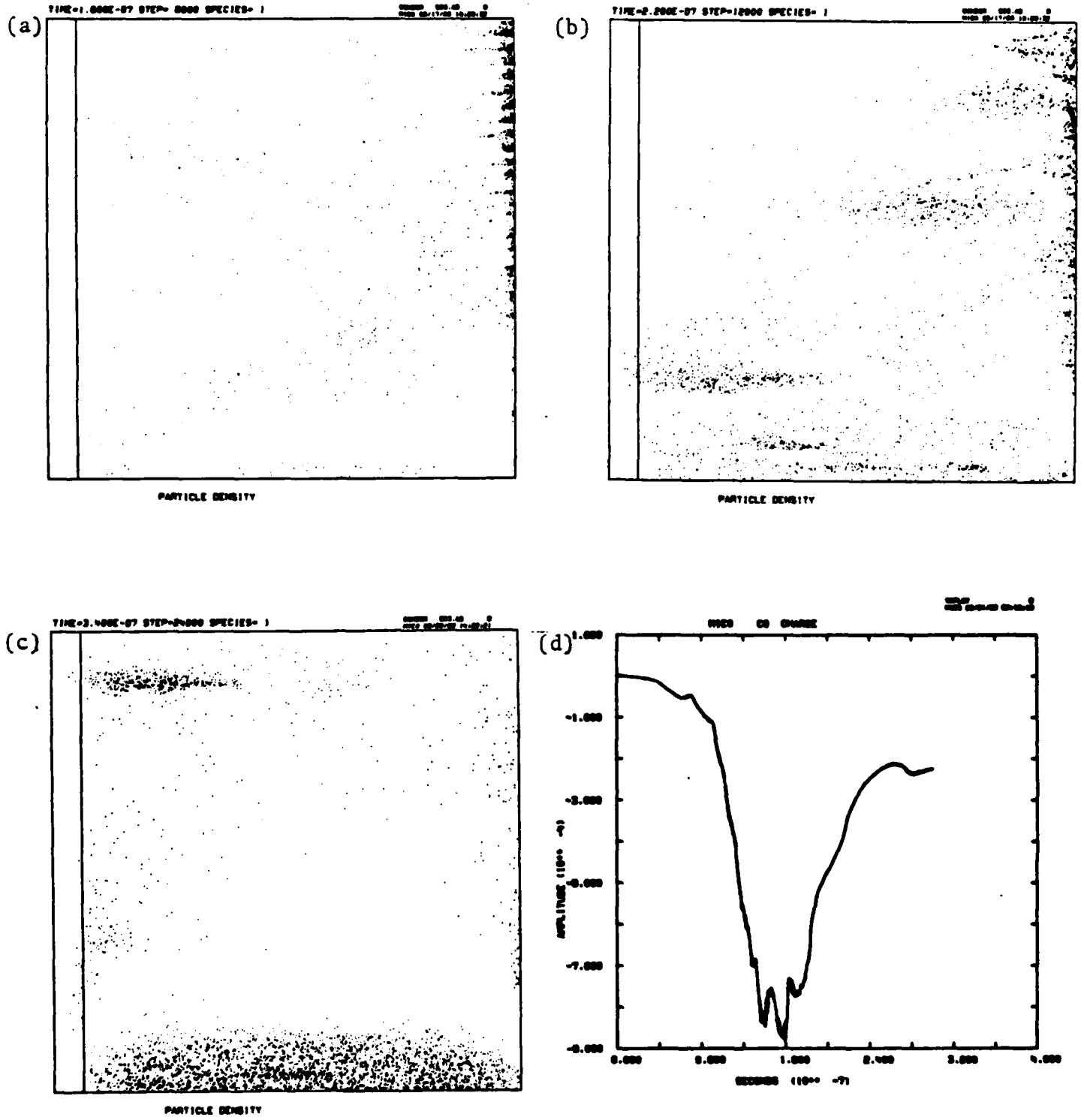


Fig. 4-13: Particle Density (r-z plots) and Total Charge on the Grid vs. time.

(a) Particle Density at t=160 ns
(b) Particle Density at t=220 ns
(c) Particle Density at t=340 ns
(d) Total Charge on Grid.

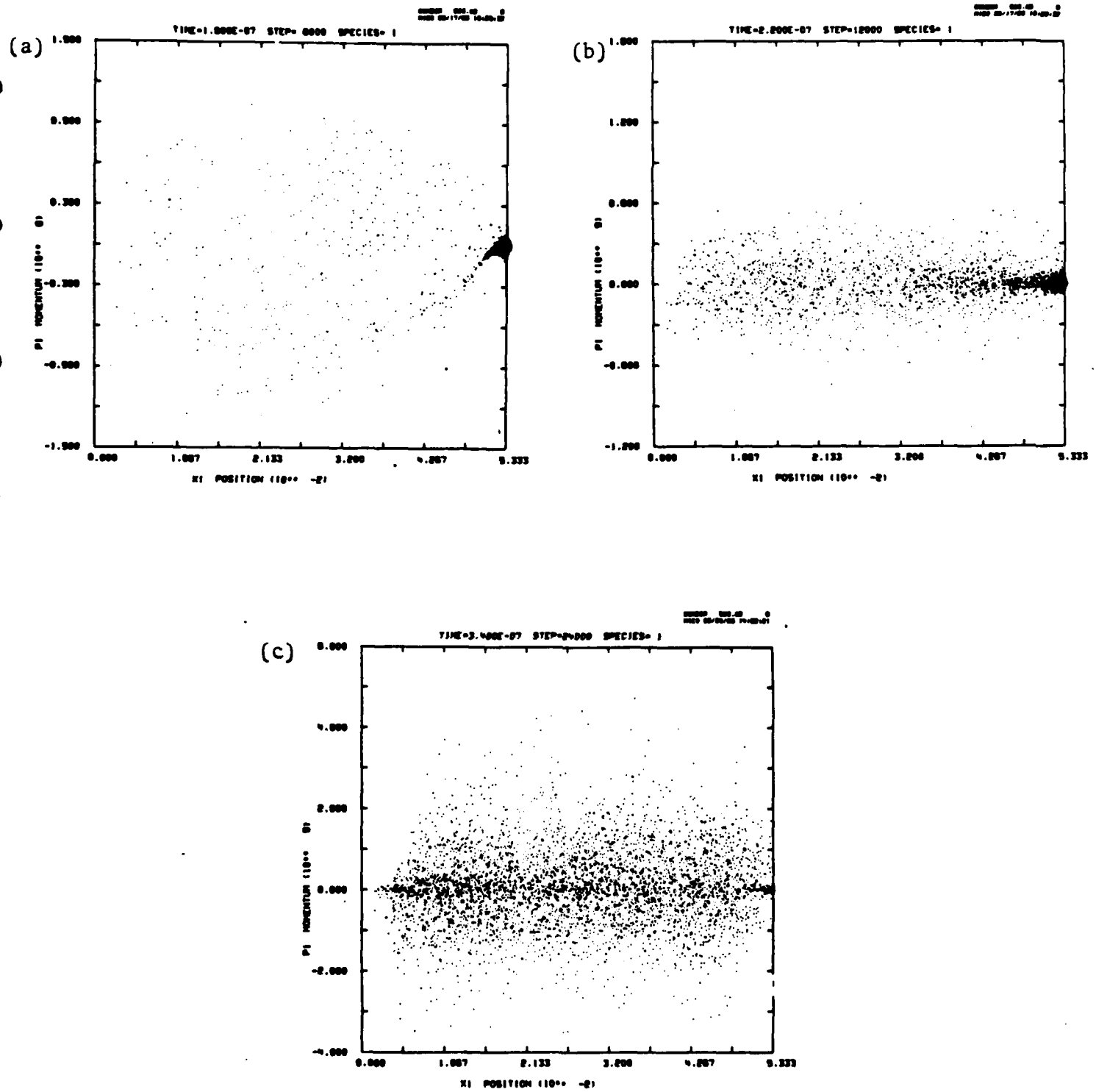


Fig. 4-14: Axial Momentum, P_z, vs. Z.

(a) t=160ns; (b) t=220ns; (c) t=340ns.

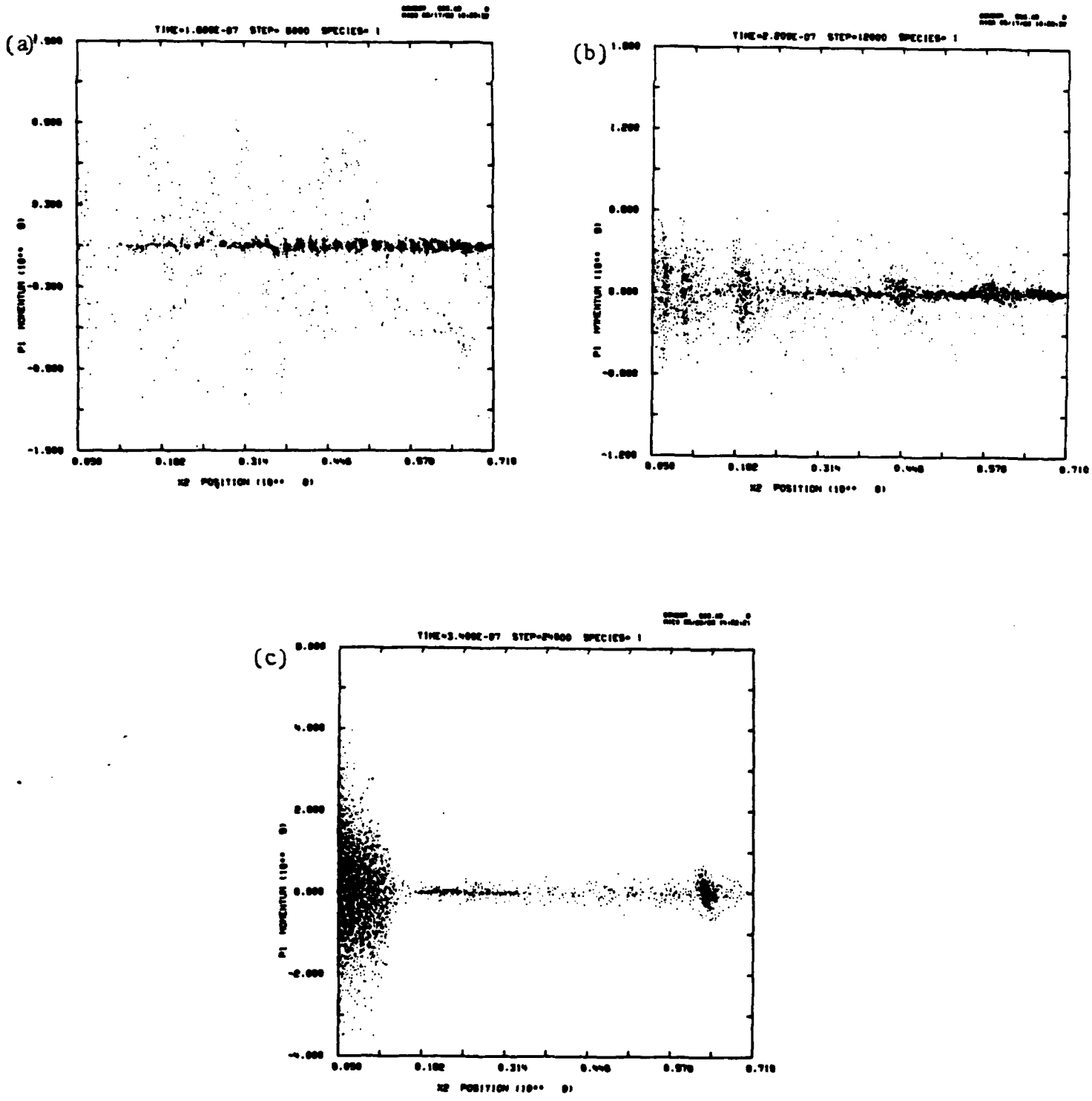


Fig. 4-15: Axial Momentum P_z, vs. r.
(a) t=160ns; (b) t=220ns; (c) t=340ns.

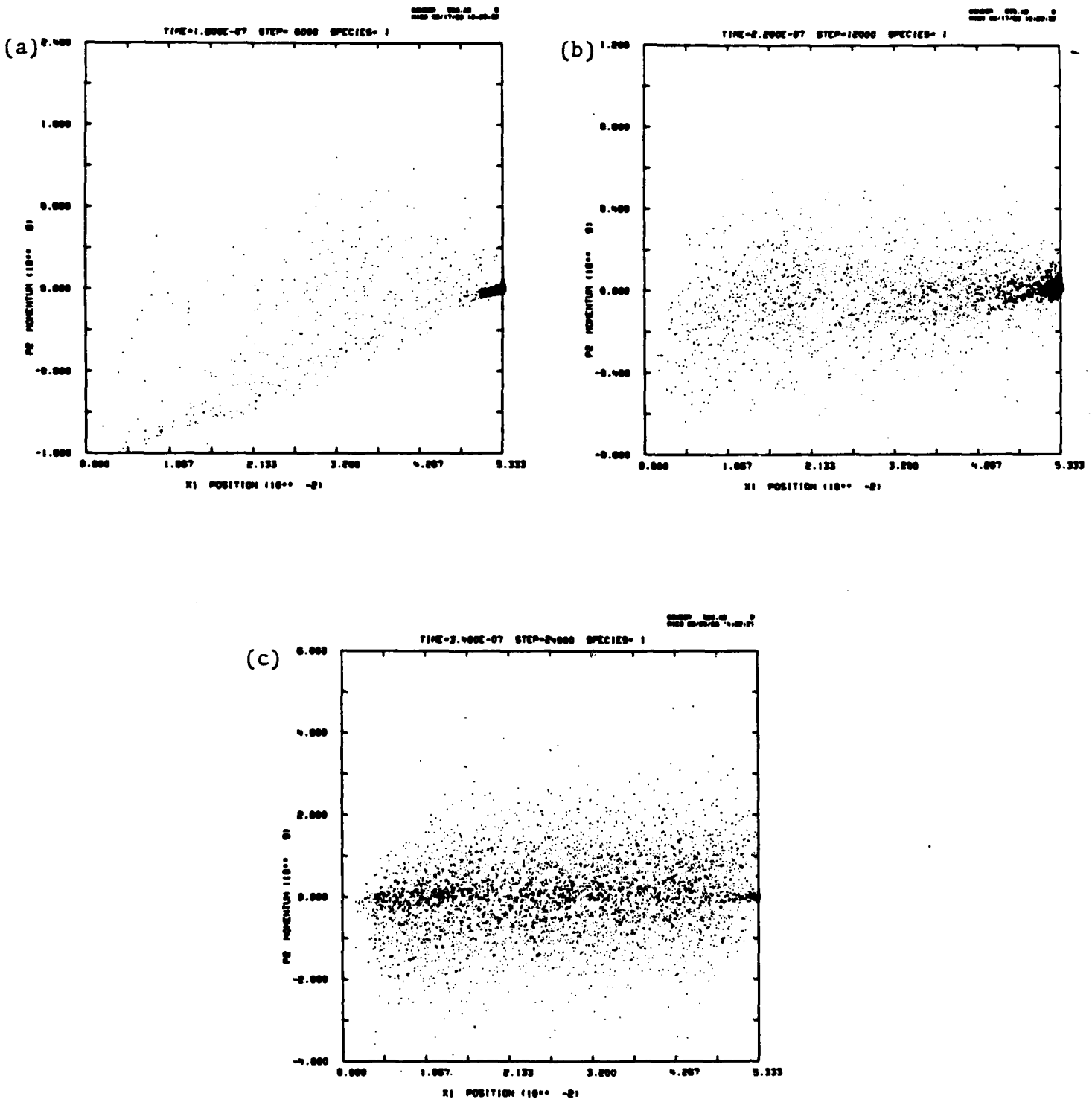


Fig. 4-16: Radial Momentum, P_r, vs. Z.

(a) t=160ns; (b) t=220ns; (c) t=340ns.

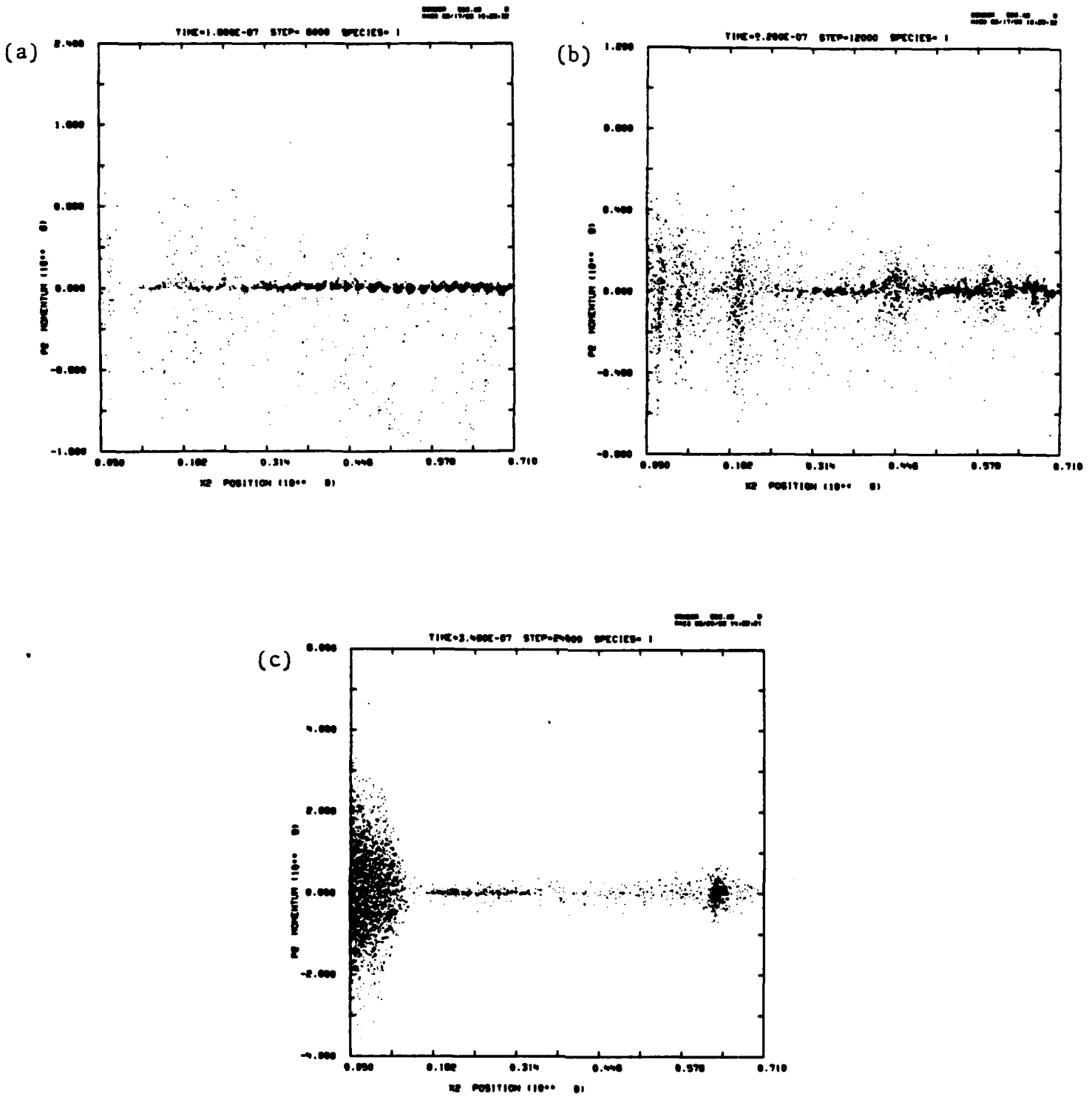


Fig. 4-17: Radial Momentum, P_r, vs. r.
(a) t=160ns; (b) t=220ns; (c) t=340ns.

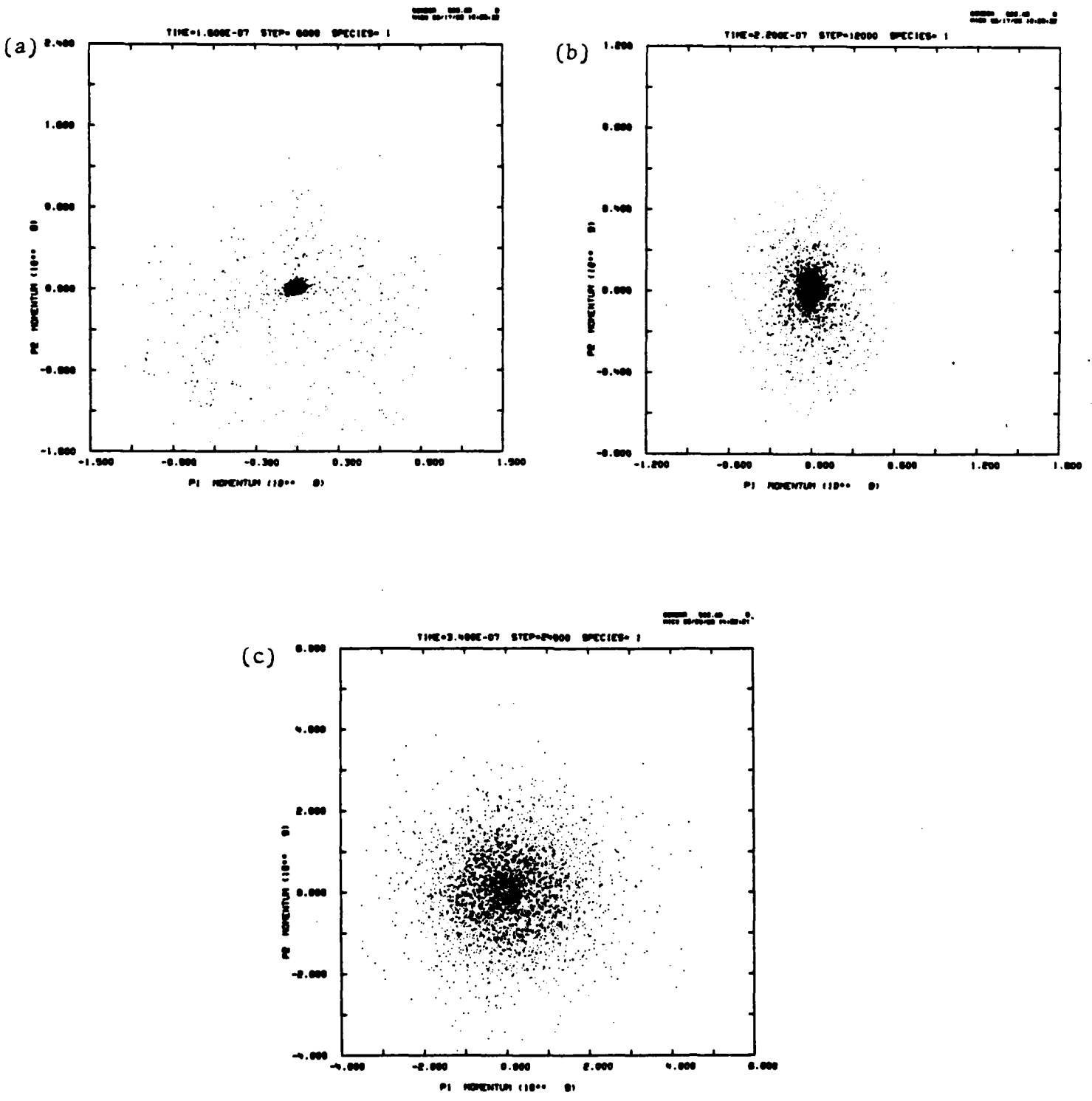


Fig. 4-18: Momentum Plane (P_r vs. P_z).
(a) t=160ns; (b) t=220ns; (c) t=340ns.

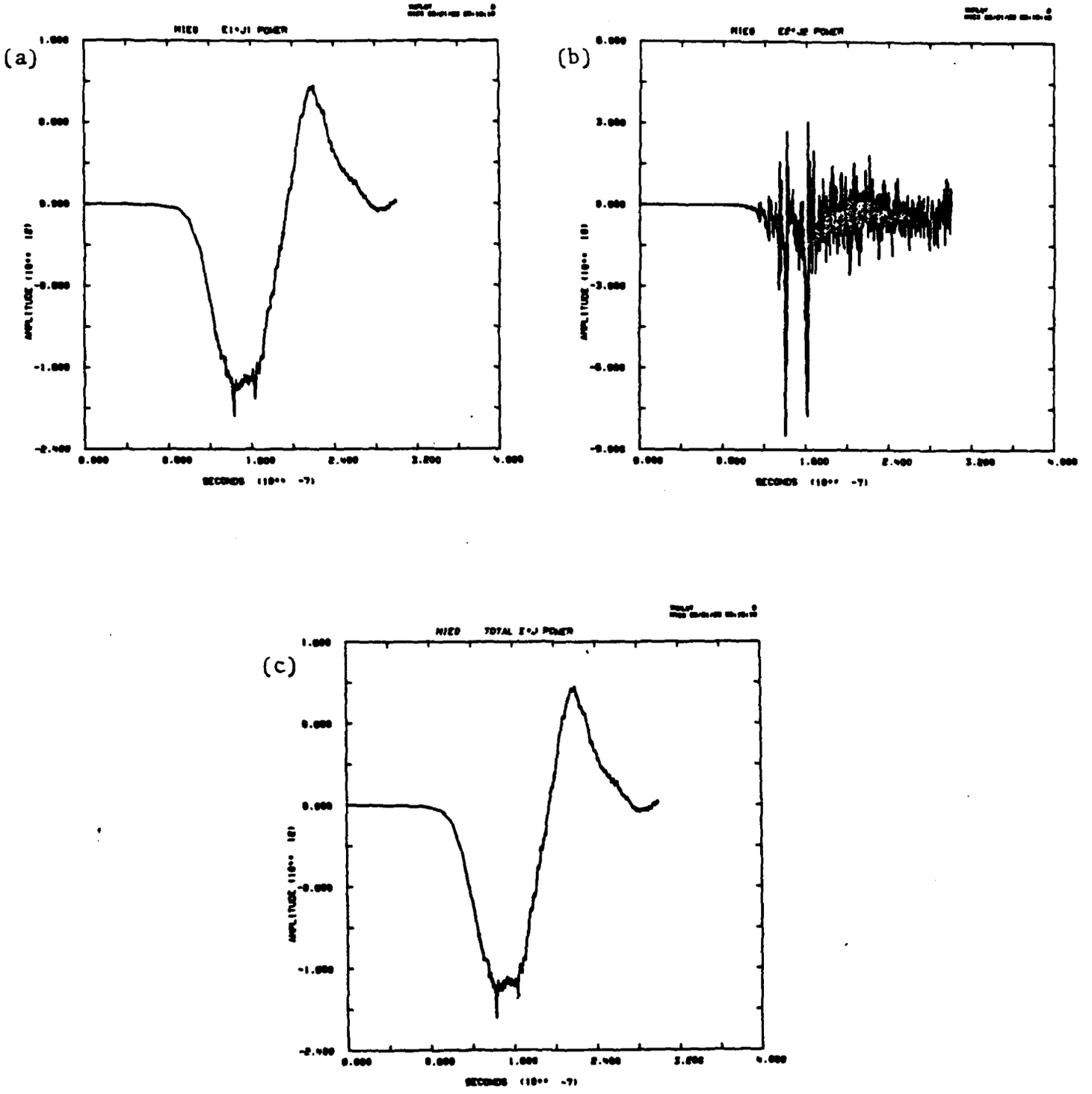


Fig. 4-19: $\underline{E} \cdot \underline{J}$ Power vs. time.

(a) $E_z J_z$ Component; (b) $E_r J_r$ Component;
(c) Total $\underline{E} \cdot \underline{J}$ Power.

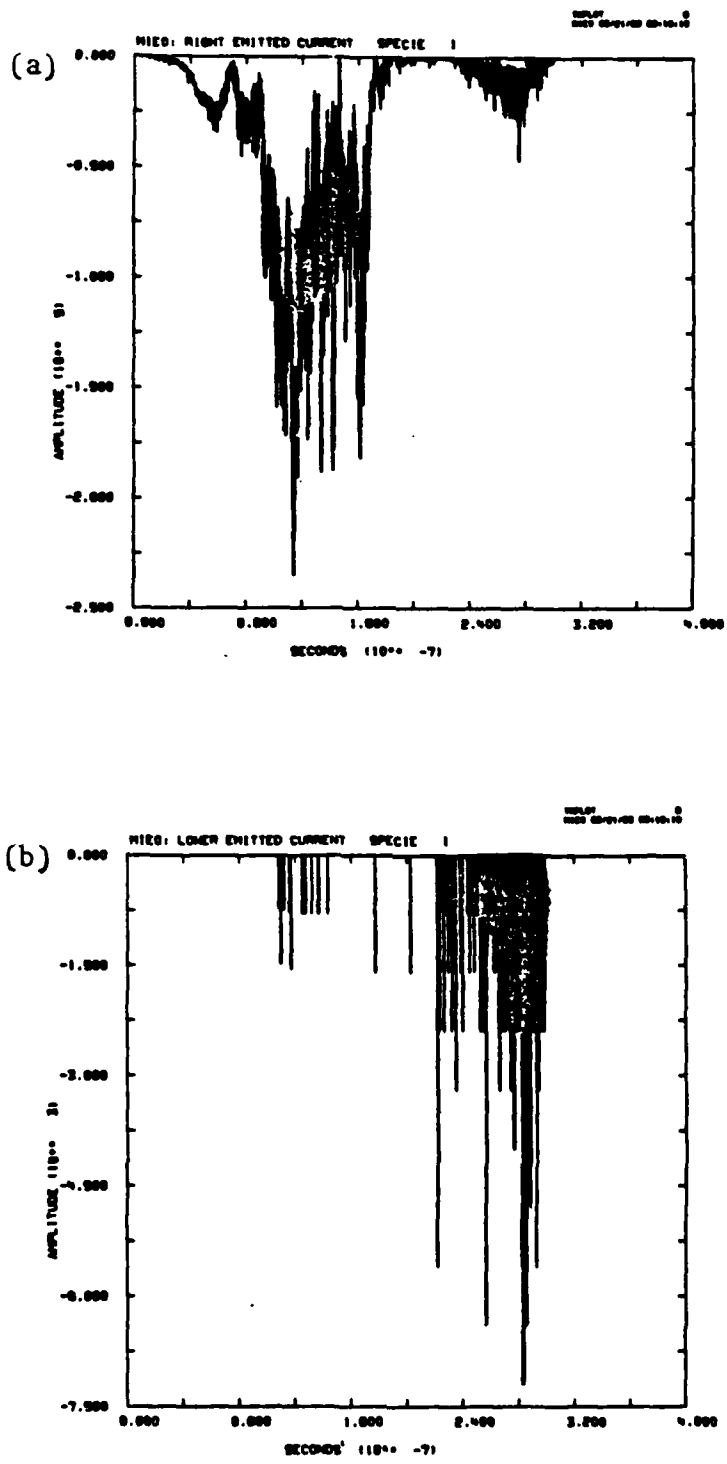


Fig. 4-20: Emitted Current vs. time.

(a) Current emitted from the cathode (right) surface:

(b) Current emitted from the short-circuit rod(lower) surface.

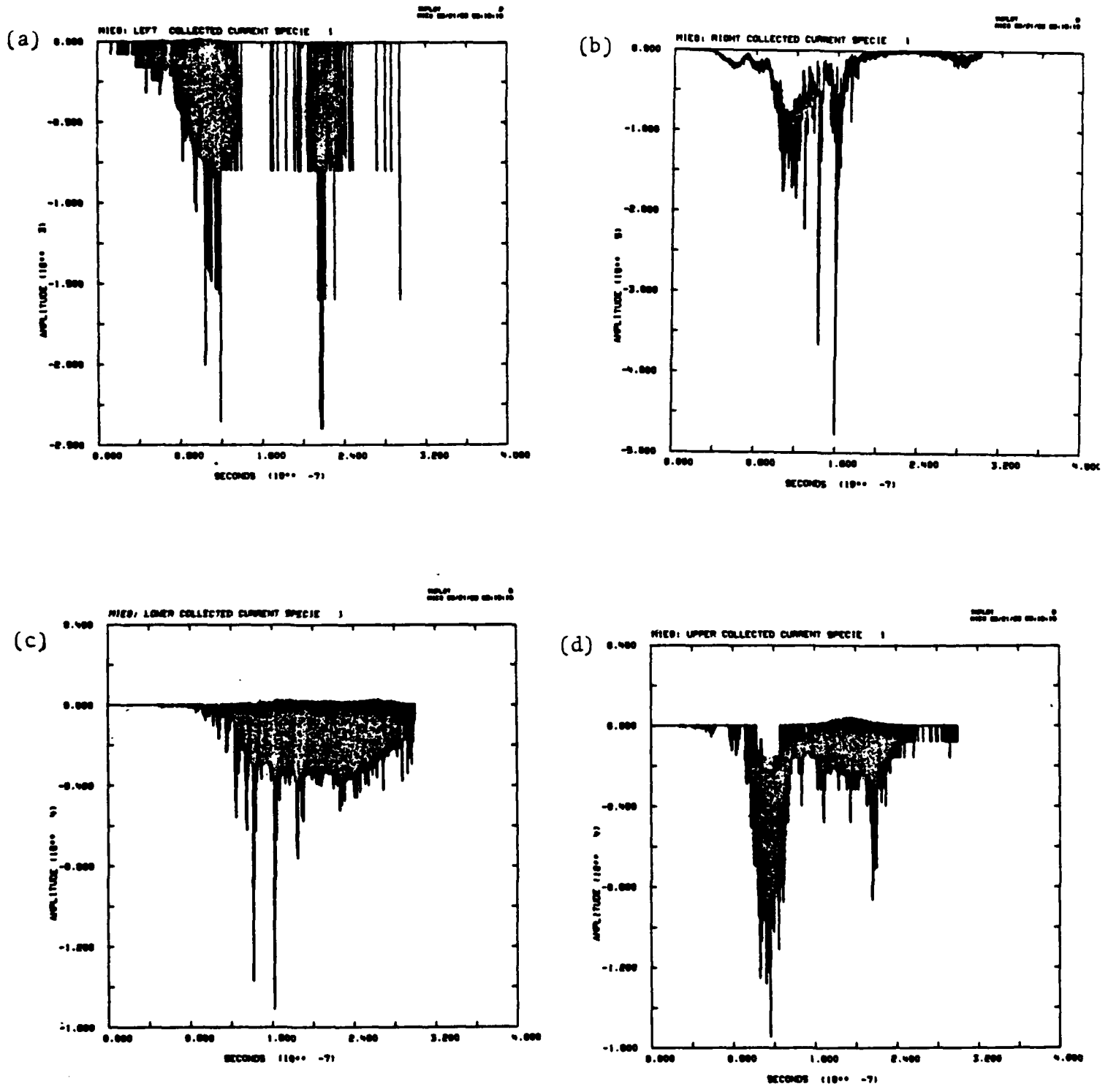


Fig. 4-21: Collected current vs. time.

(a) Current collected on left surface;
(b) Current collected on right surface;
(c) Current collected on lower surface;
(d) Current collected on upper surface.

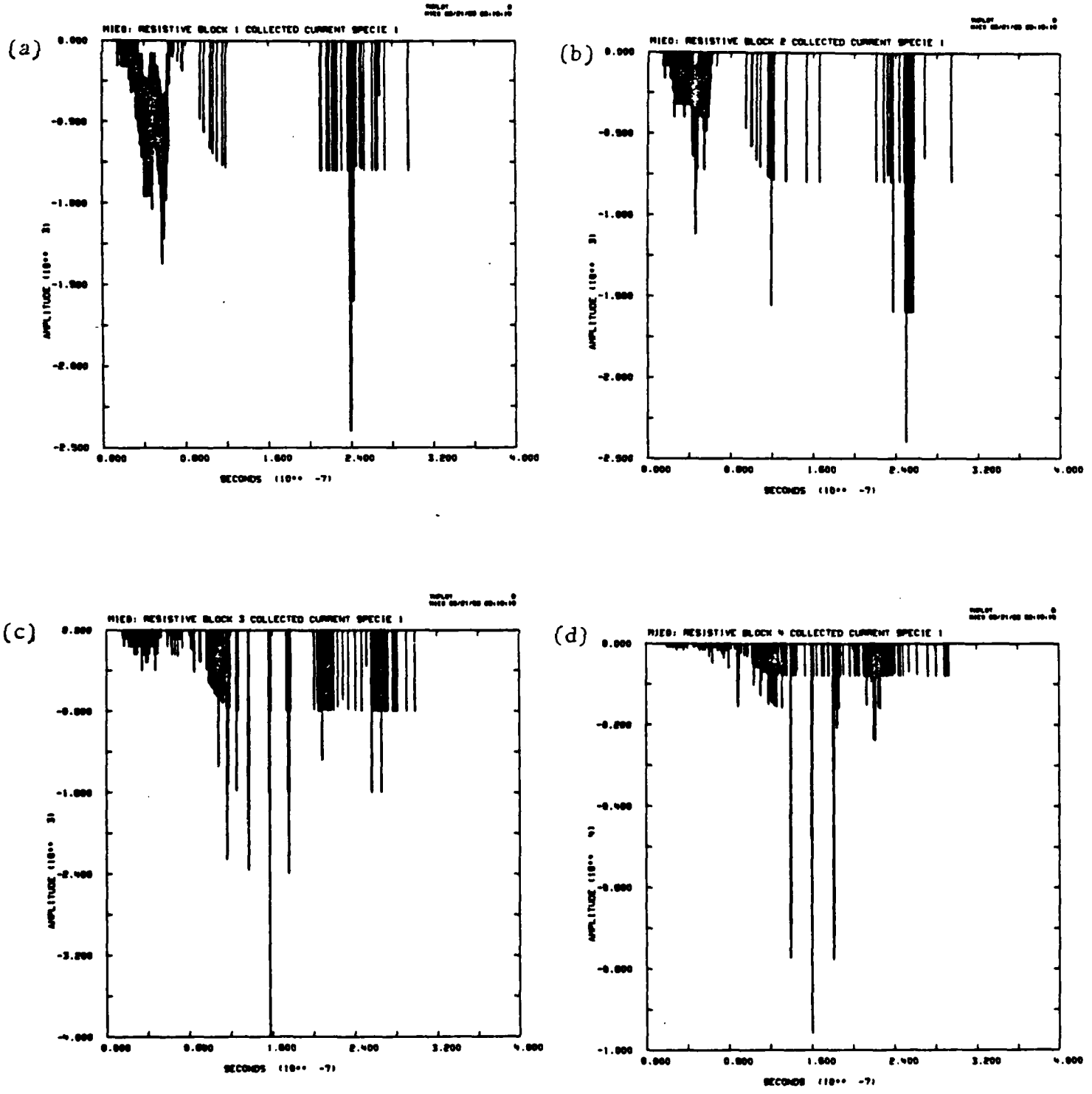


Fig. 4-22: Current Collected on Faraday cups (absorbers) vs. time.

(a) Faraday cup No. 1
(b) Faraday cup No. 2
(c) Faraday cup No. 3
(d) Faraday cup No. 4

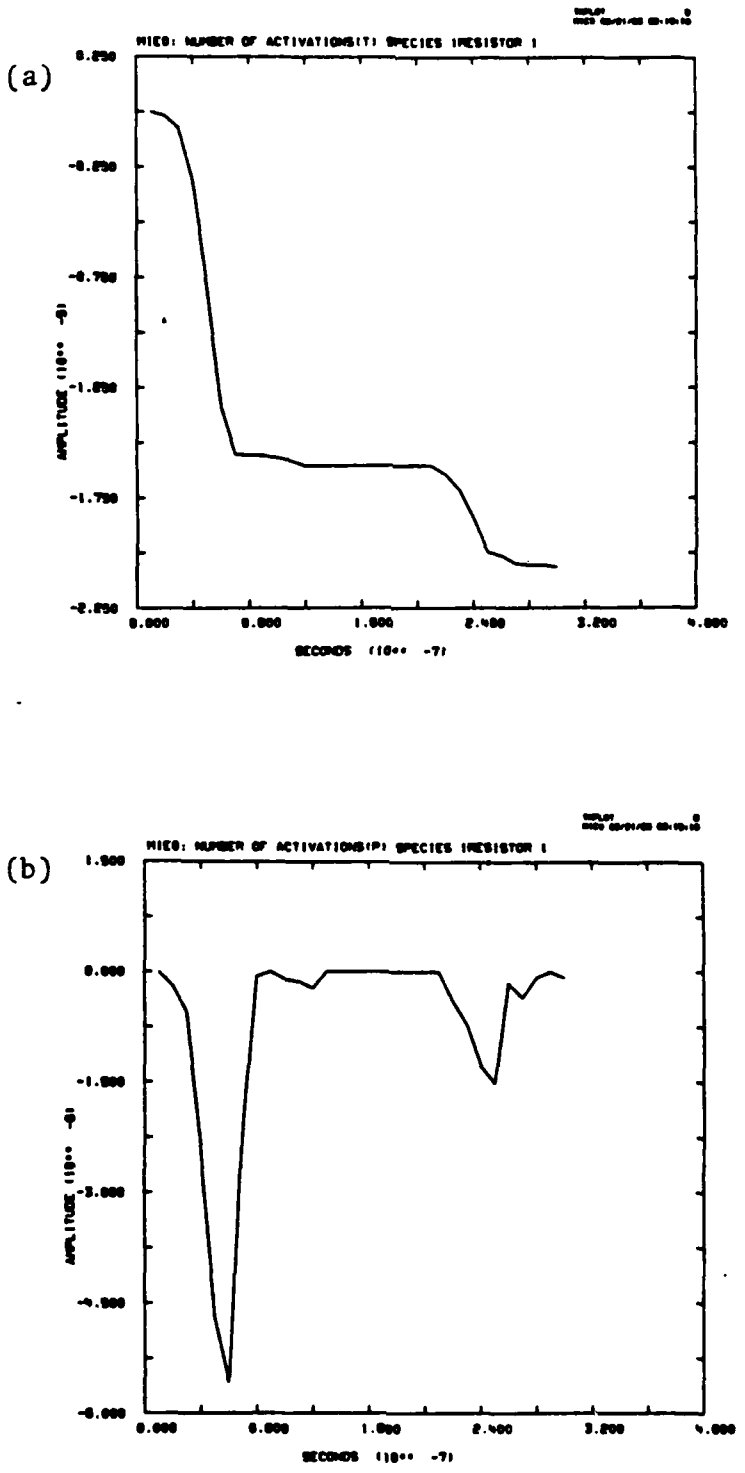


Fig. 4-23: Integrated charge collection in Faraday Cup No. 1 vs. time.

(a) Total integrated charge collected to time t;
(b) Charge collected over 10ns prior to time t.

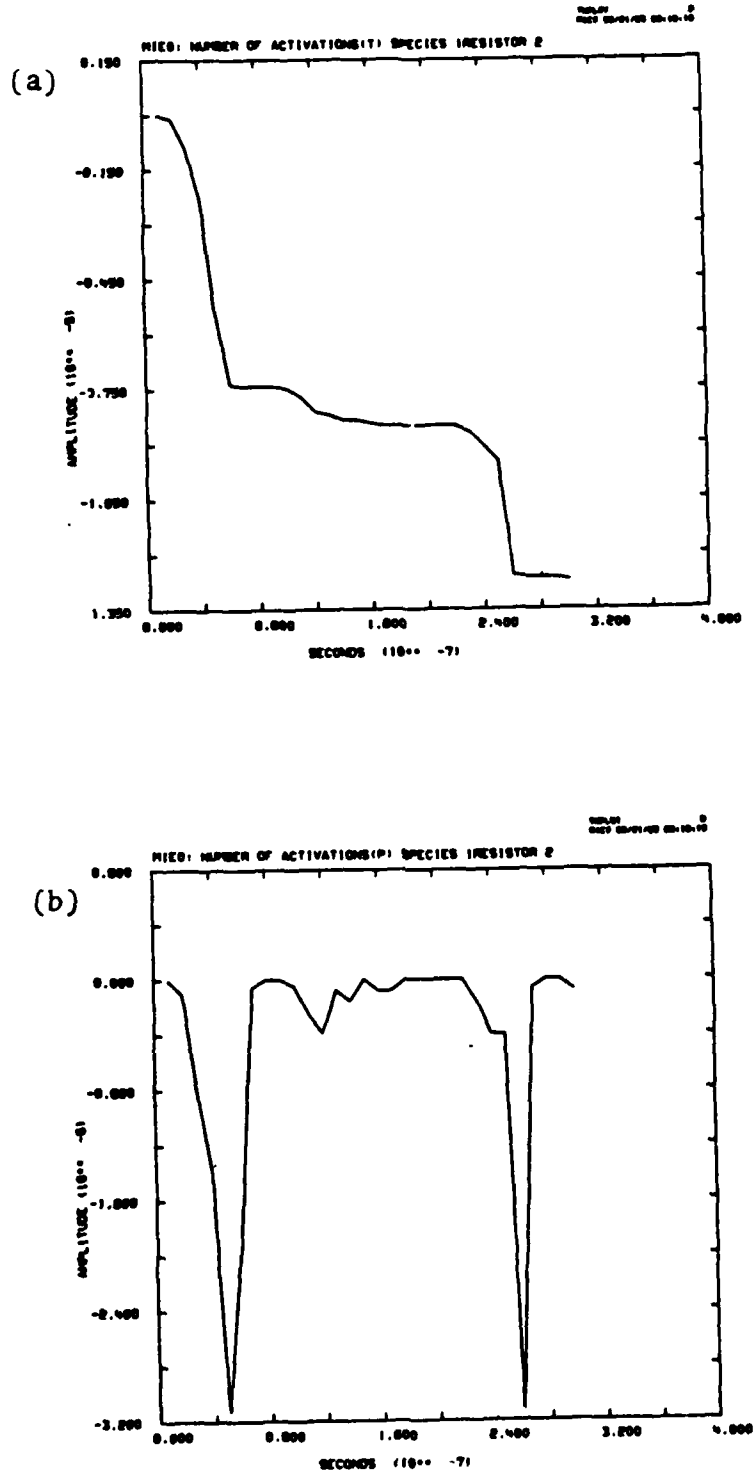


Fig. 4-24: Integrated charge collection in Faraday cup No. 2 vs. time.

(a) Total charge collected to time t;
(b) Charge collected over 10ns prior to t.

Appendix A

WIRES CODE

The WIRES code, based on the model described in Section 2, is basically a Runge-Kutta integrator for five variables:

(1) array radius,

(2) implosion speed,

(3) current,

(4) radiation yield in photons with energy greater than ε^* (an input),

(5) total radiation yield.

If run interactively, the code will prompt the user for the following data:

Block 1

N = number of wires
EST= spectrum cut-off energy (eV)
XMU= single wire mass per unit length (g/cm)
XL = wire length (cm)

Block 2

R(Ø) = initial array radius (cm)
B = outer radius for return current
Z = atomic number of wire material
XMASS = atomic mass (amu) of wire material
CLOG = Coulomb logarithm (default value = 4)
GAMMA = specific heat ratio (default value = 5/3)
EMISS1= emissivity for $h\nu$<EST (default value = $5.\times 10^{-4}$)
EMISS2= emissibity for $h\nu$>EST (default value = $5.\times 10^{-6}$)
NPFLAG= (1 or 0) = (Yes or No) print during integration for wire cooling after assembly.

Block 3

VØ = circuit charge voltage (Volts)
ZØ = generator impedance (Ohms)
XLD= diode inductance (Henries)

Block 4

DT = time step for Runge-Kutta (sec)
NPRINT = number of time steps between print-out's.

Each data block should be entered in free-format as a single-line input.

The main program initializes the problem and calls the following subroutines:

(1) STEP: Calculates one Runge-Kutta time step, using subroutine FORCE to calculate the necessary first derivatives.

(2) FORCE: Provides derivatives for use by STEP. FORCE finds the temperature by imposing a local Bennett equibrium,

$$\frac{B^2}{8\pi} = n\ (1+Z_{eff})K_B T.$$

(3) RADFRAC: Calculates the fraction of the black-body radiation yield which lies above EST.

(4) XCURR: Allows a specified current waveform to be utilized; this option is not used in the current version of WIRES.

(5) OUT: Print-out subroutine. The following quantities are printed:

(a) T = time
(b) Y(1) = array radius
(c) Y(2) = implosion speed
(d) Y(3) = current
(e) Y(4) = yield for $h\nu$>EST
(f) Y(5) = total yield
(g) A = wire radius
(h) T(EV) = wire temperature (eV)
(i) ZEFF = effective ionization state
(j) DENS = number density
(k) RP(OHMS) = wire resistance(Spitzer)
(l) LP(H) = wire inductance (Russell)
(m) VI(W) = input power = IV
(n) P-OHMIC = I^2Rp = Ohmic dissipation
(o) P-BB = blackbody radiation power

(p) W-FLD = $LI^2/2$ = stored field energy
(q) W-KIN = $N\mu\ell v^2/2$ = kinetic energy
(r) W-INT = $1.5\ n(1+Z_{eff})\ K_B T$ = internal energy

(6) XVOLTS: specifies applied voltage waveform.

(7) FINAL: Calculates final assembly and cooling of plasma cylinder, based on instantaneous conversion of kinetic energy to temperature followed by radiative cooling via blackbody emission.

(8) DERIV: Provides derivatives for

(a) temperature
(b) yield above EST
(c) total radiation yield

needed by subroutine FINAL.

Appendix B

EGVPRB CODE

The EGVPRB Code, together with pre-processors (EGVSETUP and MHDEQUIL) and post-processors (EGVPLT and EGCOPLT) are described in an on-line documentation file, EGVPRB.INF, which is listed below.

The various modules are

(1) EGVSETUP : File assignment.

(2) MHDEQUIL.FOR : MHD equilibrium specification.

(3) EGVPRB.FOR : Linear, ideal MHD stability analysis.

(4) EGVPLT.FOR : Plots coefficients and eigenfunction for converged solution.

(5) EGCOPLT.FOR : Plots coefficients and eigenfunctions for sequence of trial solutions.

The main module, EGVPRB.FOR, solves general second-order, differential eigenvalue problems of the form,

$$a\xi'' + b\xi' + c\xi = 0,$$

where $\xi(r)$ is the eigenfunction, and the coefficients, $a(r,\lambda)$, $b(r,\lambda)$, and $c(r,\lambda)$, are functions of both r and the eigenvalue parameter, λ. This code, specialized to solve the linear, ideal MHD stability problem for a specified cylindrical equilibrium, is set-up on the JAYCOR VAX Computer (Host CAIN, Directory [IPR3]). The MHD stability analysis itself is described in Section 3.

The major subunits of the EGVPRB code are the following:

(1) SYSODE: the main program, a top-level governor.

(2) MATRIX: reads input data and calculates the a,b,c coefficients.

(3) FCN: calculates the determinant which provides the characteristic equation for the eigenvalue.

(4) CEVALF: a root finder which solves the characteristic equation for the eigenvalue.

(5) BC: a subroutine which sets-up the specified boundary conditions.

(6) MATNRM: normalizes the determinant to avoid overflow/underflow.

(7) NRMFCN: eigenfunction calculation

(8) PLTFCN: sets-up output plots

(9) DEPSE: decomposition of function into B-splines.

(10) REPSE: recomposition of function from B-splines.

(11) REPSP: recomposition of first derivative of function from B-splines.

The EGVPRB package was designed for the CRAY computer system, and several CRAY-dependent lines of code were "commented-out" of the code to adapt it to the VAX. The unfortunate overflow/underflow limits on the VAX (approximately $10^{\pm 35}$) impose a limitation on the number of knots (or nodes) which may be carried in the splines. The determinant to be computed is an $N \times N$ determinant, where N is the number of knots. While the determinant is normalized, the VAX can overflow or underflow easily if $N \geq 30$ is utilized. An

input parameter, SETNRM, has been built into the code to "fine tune" the determinant normalization so as to avoid this problem. With SETNRM specified as 1, the code normalizes the determinant to the largest element in the matrix. Test problems using N=20 have run without difficultly with SETNRM=1.

Listings of the various modules follow.

APPENDIX C

Listing of WIRES

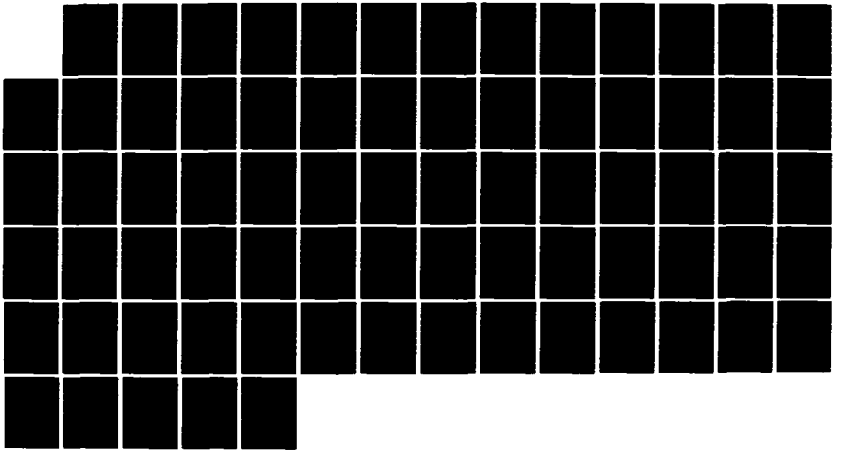
```
      PROGRAM WIRES
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(NDIM=5)
      DIMENSION Y(NDIM),DY(4,NDIM),YOLD(NDIM)
      COMMON N,EST,XMU,RHO,XL,B,Z,CLOG,NPFLAG,
     * XMASS,GAMMA,DT,TEMP,A,EMISS1,EMISS2
      COMMON/CIRC/VO,ZO,XLD,XLDOT,XLP,RP,ZEFF
      DATA PI/3.14159265358979238D0/
C
C       PROGRAM TO CALCULATE IMPLOSION
C       OF WIRE ARRAYS
C      Y(1)=ARRAY RADIUS
C      Y(2)=IMPLOSION SPEED
C      Y(3)=CURRENT
C      Y(4)=YIELD ABOVE EST
C      Y(5)=TOTAL YIELD
C
C       INPUTS
C      N=NUMBER OF WIRES
C      EST=CUT-OFF ENERGY (EV)
C      XMU=WIRE MASS/LENGTH
C      XL=WIRE LENGTH (CM)
C      R(0)=INITIAL ARRAY RADIUS (CM) =Y(1)
C      B=OUTER RADIUS FOR RETURN CURRENT (CM)
C      Z=ATOMIC NUMBER
C      CLOG=COULOMB LOG (DEF:4)
C      XMASS=ATOMIC MASS (AMU)
C      GAMMA=SPECIFIC HEAT RATIO (DEF:5/3)
C      EMISS1=EMISSIVITY BELOW EST (DEF: 5.E-4)
C      EMISS2=EMISSIVITY ABOVE EST (DEF: 5.E-6)
C      NPFLAG=(1,0)=(YES,NO) PRINT DURING TEMP DECAY
C      VO=CIRCUIT CHARGE VOLTAGE
C      ZO=GENERATOR IMPEDANCE
C      XLD=DIODE INDUCTANCE
C      DT=TIME STEP (SEC)
C      NPRINT=INTERVAL BETWEEN PRINTS
C
      GAMMA=5./3.
      CLOG=4.
      EMISS1=5.E-4
      EMISS2=5.E-6
      Y(2)=0.
      Y(3)=1.E5
      Y(4)=0.
      Y(5)=0.
      IFIRST=0
      PRINT 900
900   FORMAT(1X,'N,EST(EV),XMU,XL'/
     * 1X,'R(0),B,Z,XMASS(AMU),CLOG,GAMMA,',
     *'EMISS1,EMISS2,NPFLAG'/
     * 5X,'NPFLAG=1 FOR PRINT'/
     * 1X,'VO(VOLTS),ZO(OHMS),XLD(HENRIES)'/
     * 1X,'DT,NPRINT')
      READ(5,*) N,EST,XMU,XL
      READ(5,*) Y(1),B,Z,XMASS,CLOG,GAMMA,EMISS1,EMISS2,NPFLAG
      READ(5,*) VO,ZO,XLD
      READ(5,*) DT,NPRINT
      T=0.
      KOUNT=0
      KPRINT=0
10    CALL STEP(T,Y,NDIM,DY,YOLD)
      KOUNT=KOUNT+1
```

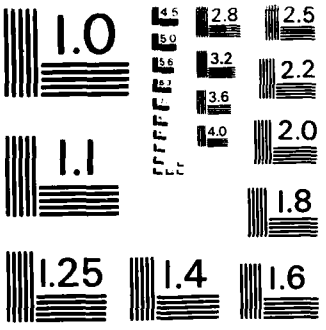
```
      KPRINT=KPRINT+1
      XN=N
      IF((IFIRST.EQ.0).AND.(A.GT.Y(1)*SIN(PI/XN))) GOTO 15
      IFIRST=1
      IF(ABS(A-Y(1)*SIN(PI/XN)).LT.1.E-3*A) GOTO 20
      DT=MIN(DT,.5*(A/SIN(PI/XN)-Y(1))/Y(2))
15    IF(KPRINT.LT.NPRINT) GOTO 10
      KPRINT=0
      CALL OUT (T,Y,NDIM)
      GOTO 10
20    CALL OUT(T,Y,NDIM)
      CALL FINAL(T,Y,NDIM)
      STOP
      END
      SUBROUTINE STEP(T,Y,NDIM,DY,YOLD)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(NDIM),DY(4,NDIM),YOLD(NDIM),D(4)
      COMMON N,EST,XMU,RHO,XL,B,Z,CLOG,NPFLAG,
     * XMASS,GAMMA,DT,TEMP,A,EMISS1,EMISS2
      TOLD=T
      DO 5 I=1,NDIM
5     YOLD(I)=Y(I)
      D(1)=DT/2.
      D(2)=DT/2.
      D(3)=DT
      D(4)=DT/6.
      L=1
10    CALL FORCE(T,Y,DY,NDIM,L)
      L=L+1
      IF(L.EQ.5) GOTO 20
      T=TOLD+D(L-1)
      DO 15 J=1,NDIM
15    Y(J)=YOLD(J)+DY(L-1,J)*D(L-1)
      GOTO 10
20    DO 25 J=1,NDIM
25    Y(J)=YOLD(J)+D(4)*(DY(1,J)+2.*DY(2,J)
     * +2.*DY(3,J)+DY(4,J))
      RETURN
      END
      SUBROUTINE FORCE(T,Y,DY,NDIM,LRK)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(NDIM),DY(4,NDIM)
      COMMON N,EST,XMU,RHO,XL,B,Z,CLOG,NPFLAG,
     * XMASS,GAMMA,DT,TEMP,A,EMISS1,EMISS2
      COMMON/CIRC/VO,ZO,XLD,XLDOT,XLP,RP,ZEFF
      DATA PI/3.14159265358979238D0/
      DATA SIG/5.6696E-5/,XKB/1.3807E-16/
      DATA TO/1.16E7/,XMP/1.6606E-24/
      Z26=(26./Z)**2
      XN=N
      XIP=Y(3)
      C=XIP*XIP*XMASS*XMP/200./XN**2/XMU/XKB
      TU=C
      TL=C/(1.+Z)
      DO 10 I=1,20
      TEMP=.5*(TU+TL)
      ZEFF=26.*SQRT(TEMP/(TO+Z26*TEMP))
      CTEST=TEMP*(ZEFF+1.)
      IF(CTEST.LT.C) GOTO 5
      TU=TEMP
      GOTO 10
5     TL=TEMP
10    CONTINUE
```

```
      IF(ZEFF.LT.2) GE=.582+.101*(ZEFF-1)
      IF((ZEFF.LT.4).AND.(ZEFF.GE.2)) GE=.683+.051*(ZEFF-2)
      IF((ZEFF.LT.16).AND.(ZEFF.GE.4))GE=.785+.0115*(ZEFF-4)
      IF(ZEFF.GE.16) GE=1.-1.232/ZEFF
      RHO=3800.*ZEFF*CLOG/GE/TEMP**(1.5)
      CALL RADFRAC(EST,TEMP,FRAC)
      EMISS=EMISS1*(1.-FRAC)+EMISS2*FRAC
      A=(1.E7*RHO*XIP*XIP/2./PI/PI/XN/XN/SIG/EMISS/TEMP**4)**(1./3.)
      RP=RHO*XL/PI/A**2/XN
      XLP=XL*(.5+2.*LOG(B**N/XN/A/Y(1)**(N-1)))/XN*1.E-9
      XLDOT=-2.*XL*(XN-1.)*Y(2)/Y(1)/XN*1.E-9
      CALL XVOLTS(T,V)
      AS=2.*PI*A*XL*XN
      DY(LRK,1)=Y(2)
      DY(LRK,2)=-(XN-1.)*(XIP/10./XN)**2/XMU/Y(1)
      DY(LRK,3)=(V-(ZO+RP+XLDOT)*XIP)/(XLD+XLP)
      DY(LRK,4)=FRAC*SIG*EMISS2*AS*TEMP**4
      DY(LRK,5)=SIG*EMISS*AS*TEMP**4
      RETURN
      END
      SUBROUTINE RADFRAC(EST,TEMP,FRAC)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION YLT(70),YFRAC(70)
      DATA YLT/.01,.02,.03,.04,.05,.055,.06,
     * .065,.07,.075,.08,.085,.09,.095,
     * .10,.11,.12,.13,.14,.15,.16,.17,.18,.19,
     * .20,.22,.24,.26,.28,.30,.32,.34,.36,.38,
     * .40,.45,.50,.55,.60,.65,.7,.8,.9,1.,
     * 1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.,
     * 2.5,3.,3.5,4.,5.,6.,7.,8.,9.,10.,15.,
     * 20.,30.,40.,50.,100./
      DATA YFRAC/0.,3.7E-27,2.7E-17,1.9E-12,
     * 1.3E-9,1.35E-8,9.29E-8,4.67E-7,1.84E-6,
     * 5.94E-6,1.64E-5,3.99E-5,8.7E-5,1.73E-4,
     * 3.21E-4,9.11E-4,.00213,.00432,.00779,
     * .01285,.01971,.02853,.03933,.05210,
     * .06672,.10087,.14024,.18310,.22787,
     * .27320,.31807,.36170,.40327,.44334,
     * .48084,.56428,.63370,.69086,.73777,
     * .77630,.80806,.85624,.88998,.91415,
     * .93184,.94505,.95509,.96285,.96893,
     * .97376,.97765,.98081,.98340,.98555,
     * .99216,.99529,.99695,.99792,.99890,
     * .99935,.99959,.99972,.99980,.99985,
     * .999955,.99998,.9999943,.9999975,.9999988,
     * .99999985/
      DATA HC/1.2399E-4/,C2/1.43883/
      XLT=TEMP*HC/EST
      IF(XLT.GE..02) GOTO 10
      FRAC=0.
      RETURN
10    IF(XLT.LE.100.) GOTO 20
      X=C2/XLT
      FRAC=1.-.0513*X**3
      RETURN
20    DO 30 I=2,70
      IF(XLT.GT.YLT(I)) GOTO 30
      FRAC=YFRAC(I-1)+(YFRAC(I)-YFRAC(I-1))*
     * (XLT-YLT(I-1))/(YLT(I)-YLT(I-1))
      GOTO 40
30    CONTINUE
40    RETURN
      END
```

```
      SUBROUTINE XCURR(XIP,T)
      IMPLICIT REAL*8(A-H,O-Z)
      XIP=5.E6
      RETURN
      END
      SUBROUTINE OUT(T,Y,NDIM)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(NDIM)
      COMMON N,EST,XMU,RHO,XL,B,Z,CLOG,NPFLAG,
     * XMASS,GAMMA,DT,TEMP,A,EMISS1,EMISS2
      COMMON/CIRC/VO,ZO,XLD,XLDOT,XLP,RP,ZEFF
      DATA PI/3.14159265358979323BD0/,XMP/1.6606E-24/
      DATA SIG/5.6696E-5/,XKB/1.3807E-16/
      WRITE(6,999)
  999 FORMAT(19X,'T',11X,'Y(1)',11X,'Y(2)',11X,'Y(3)',
     * 11X,'Y(4)',11X,'Y(5)')
  998 FORMAT(34X,'A',10X,'T(EV)',11X,'ZEFF',11X,'DENS')
  997 FORMAT(27X,'RP(OHMS)',10X,'LP(H)',10X,'VI(W)')
  996 FORMAT(28X,'P-OHMIC',11X,'P-BB',10X,'P-KIN')
  995 FORMAT(30X,'W-FLD',10X,'W-KIN',10X,'W-INT')
      TTTT=TEMP/11600.
      WRITE(6,1000) T,(Y(I),I=1,NDIM)
      WRITE(6,998)
      DENS=XMU/PI/A**2/XMASS/XMP
      CALL XVOLTS(T,V)
      CALL RADFRAC(EST,TEMP,FRAC)
      EMISS=EMISS1*(1.-FRAC)+EMISS2*FRAC
      VI=V*Y(3)
      XI2R=RP*Y(3)**2
      XN=N
      AS=2.*PI*A*XL*XN
      PBB=SIG*EMISS*TEMP**4*AS*1.E-7
      XI2LD=.5*XLDOT*Y(3)**2
      WFLD=.5*(XLD+XLP)*Y(3)**2*1.E7
      WKIN=.5*XMU*XL*Y(2)**2*XN
      WINT=1.5*DENS*(ZEFF+1.)*XKB*TEMP
      WRITE(6,1001) A,TTTT,ZEFF,DENS
      WRITE(6,997)
      WRITE(6,1002) RP,XLP,VI
      WRITE(6,996)
      WRITE(6,1002) XI2R,PBB,XI2LD
      WRITE(6,995)
      WRITE(6,1003) WFLD,WKIN,WINT
 1000 FORMAT(5X,6E15.5)
 1001 FORMAT(20X,4E15.5)
 1002 FORMAT(20X,3E15.5)
 1003 FORMAT(20X,3E15.5/)
      RETURN
      END
      SUBROUTINE XVOLTS(T,V)
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON/CIRC/VO,ZO,XLD,XLDOT,XLP,RP,ZEFF
      V=VO
      RETURN
      END
      SUBROUTINE FINAL(T,Y,NDIM)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(NDIM),ZVECT(3),DZV(4,3),D(4),ZOLD(3)
      COMMON N,EST,XMU,RHO,XL,B,Z,CLOG,NPFLAG,
     * XMASS,GAMMA,DT,TEMP,A,EMISS1,EMISS2
      COMMON/CIRC/VO,ZO,XLD,XLDOT,XLP,RP,ZEFF
      DATA PI/3.14159265358979323BD0/
      DATA XMP/1.6606E-24/,TO/1.16E7/
```

AD-A139 628 PLASMA PHYSICS ISSUES IN ADVANCED SIMULTTION RESEARCH 2/2
(U) SCIENCE APPLICATIONS INC MCLEAN VA A MONDELLI
01 NOV 83 SAI-84-235-WA SBI-AD-E001 641
UNCLASSIFIED N00014-81-C-2041 F/G 20/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

```
      DATA SIG/5.6696E-5/, XKB/1.3807E-16/
C     FINAL ASSEMBLY -- CONVERT TO CYLINDER
C     VA=ALFVEN SPEED
C     XKK=1/RO=WAVENUMBER FOR MAXIMUM GROWTH
C     TASBLY=5*MHD GROWTH TIME
C     KINETIC ENERGY CONVERTED TO TEMPERATURE
C     RADIATION ASSUMED BLACK-BODY
C        ZVECT(1)=TEMPERATURE
C        ZVECT(2)=RADIATED ENERGY ABOVE EST
C        ZVECT(3)=TOTAL RADIATED ENERGY
      XN=N
      Z26=(26./Z)**2
      ROUT=A+Y(1)
      DENS=XN*XMU/PI/ROUT**2
      BOUT=Y(3)/5./ROUT
      VA=SQRT(BOUT*BOUT/4./PI/DENS)
      XKK=1./ROUT
      TASBLY=5./XKK/VA
      WKIN=.5*XN*XMU*XL*Y(2)**2
      TIN=TEMP
      C=XMASS*XMP*Y(2)**2/(3.*XKB)
      TU=TIN+C
      TL=TIN+C/(1.+Z)
      DO 10 I=1,20
      TEMP=.5*(TL+TU)
      ZEFF=26.*SQRT(TEMP/(TO+Z26*TEMP))
      CTEST=(TEMP-TIN)*(1.+ZEFF)
      IF(CTEST.LT.C) GOTO 5
      TU=TEMP
      GOTO 10
5     TL=TEMP
10    CONTINUE
      WRITE(6,998) TEMP/11600.,ZEFF
998   FORMAT(10X,'TEMP,ZEFF=',2E15.5/)
      CALL RADFRAC(EST,TEMP,FRAC)
      EMISS=EMISS1*(1.-FRAC)+EMISS2*FRAC
      AS=2.*PI*ROUT*XL
      ZVECT(1)=TEMP
      ZVECT(2)=Y(4)
      ZVECT(3)=Y(5)
      XK2=SIG*AS
      XK1=XK2*XMASS*XMP/(1.5*XN*XMU*XL*XKB)
      DT=.05*(1.+ZEFF)/XK1/TEMP**3/EMISS
      D(1)=DT/2.
      D(2)=DT/2.
      D(3)=DT
      D(4)=DT/6.
      KPRINT=0
      NMAX=1+INT(TASBLY/DT)
      NPRINT=NMAX/100+1
      DO 40 JJJ=1,NMAX
      TOLD=T
      DO 15 KKK=1,3
15    ZOLD(KKK)=ZVECT(KKK)
      L=1
20    CALL DERIV(T,ZVECT,DZV,L,XK1,XK2,Z26,EST,EMISS1,EMISS2)
      L=L+1
      IF(L.EQ.5) GOTO 30
      T=TOLD+D(L-1)
      DO 25 KKK=1,3
25    ZVECT(KKK)=ZOLD(KKK)+DZV(L-1,KKK)*D(L-1)
      GOTO 20
30    DO 35 KKK=1,3
```

```
35      ZVECT(KKK)=ZOLD(KKK)+D(4)*(DZV(1,KKK)+2.*DZV(2,KKK)
     * +2.*DZV(3,KKK)+DZV(4,KKK))
        IF(ZVECT(1).LE.1.E3) GOTO 50
        IF(NPFLAG.EQ.0) GOTO 40
        KPRINT=KPRINT+1
        IF(KPRINT.LT.NPRINT) GOTO 40
        KPRINT=0
        WRITE(6,999) T,ZVECT(1)/11600.,ZVECT(2)*1.E-7,ZVECT(3)*1.E-7
999     FORMAT(10X,'T,TEMP,WRADG,WRAD=',4E15.5)
40      CONTINUE
50      CONTINUE
        TTTT=ZVECT(1)/11600.
        WRADG=ZVECT(2)*1.E-7
        WRAD=ZVECT(3)*1.E-7
        ZEFF=26.*SQRT(ZVECT(1)/(T0+Z26*ZVECT(1)))
        WRITE(6,1000) ROUT,TASBLY,DENS,TTTT,ZEFF,WRADG,WRAD
1000    FORMAT(1H0,20X,'FINAL ASSEMBLY'/
     * 5X,'COLLAPSE RADIUS(CM)=',E15.5/
     * 5X,'ASSEMBLY TIME(SEC)=',E15.5/
     * 5X,'DENSITY(G/CC)=',E15.5/
     * 5X,'TEMPERATURE(EV)=',E15.5/
     * 5X,'ZEFF=',E15.5/
     * 5X,'RADIATION ABOVE EST (J)=',E15.5/
     * 5X,'TOTAL RADIATION(J)=',E15.5)
        RETURN
        END
        SUBROUTINE DERIV(T,ZVECT,DZV,L,XK1,XK2,Z26,EST,
     * EMISS1,EMISS2)
        IMPLICIT REAL*8(A-H,O-Z)
        DIMENSION ZVECT(3),DZV(4,3)
        DATA T0/1.16E7/
        IF(ZVECT(1).LT.0.) ZVECT(1)=0.
        TEMP=ZVECT(1)
        TFAC=T0+Z26*TEMP
        CALL RADFRAC(EST,TEMP,FRAC)
        EMISS=EMISS1*(1.-FRAC)+EMISS2*FRAC
        DZV(L,1)=-EMISS*XK1*TEMP**4/(1.+26.*SQRT(T/TFAC)*(1.+.5*T0/TFAC))
        DZV(L,2)=EMISS2*FRAC*XK2*TEMP**4
        DZV(L,3)=EMISS*XK2*TEMP**4
        RETURN
        END
$
```

APPENDIX D

Listing of EGVPRB. INF

```
*
*-------------------------------------------------------------+
*                                                             |
*       EGVPRB.INF                                            |
*       ------ ---                                            |
*                                                             |
*-------------------------------------------------------------+
*
```

This is a user-info file for using the EGVPRB package for solving eigenvalue problems of the form,

$$av'' + bv' + cv = 0 ,$$

where the coefficients, a,b,c, depend on the independent variable, r, and on the eigenvalue. The code finds the eigenvalue as well as the eigenfunction, v(r). The current version (as of 8/1/83) of EGVPRB is set up to solve the linear, ideal MHD problem for an arbitrary cylindrical equilibrium. The user may use this package to solve other eigenvalue problems of the form given above by defining new coefficients and boundary conditions in SUBROUTINE MATRIX.

The code may be run in either of two modes. In the first mode it calls COMPLEX FUNCTION CEVALF to find the roots of thh characteristic eigenvalue equation. Alternatively, the code may be used in a mode where it examines the value of the eigenvalue equation over a range of user-specified trial eigenvalues. This second mode allows the user to search manually for the root, and to examine the behavior of the coefficients as the eigenvalue parameter is varied.

STEP 1
---- -

```
@egvsetup
```

This command causes the VAX to assign names to the various files it will use or create during the run.

FILE	NAME
for003.dat	egvprb.scr
for030.dat	egvprb.dat
for015.dat	egvplt.dat
for090.dat	egcoplt.dat

The files serve the following purposes:

- egvprb.scr -- contains the detailed printed output from the run. Only an abbreviated version is sent to the user's terminal during an interactive run.
- egvprb.dat -- input data for egvprb, containing the cylindrical equilibrium parameters. This file is written by MHDEQUIL for user-specified equilibria. A separate routine which uses the output of a rad-coupled hydro code to specify the equilibrium could be used to generate this file.
- egvplt.dat -- input data for the plotting code, EGVPLT, which plots the answer found by egvprb. This file is generated by egvprb when it is used in the mode where the code finds the root.
- egcoplt.dat-- input data for the plotting code, EGCOPLT, which plots the coefficients and trial eigenvector when the egvprb code is run in its second mode. This file is created

by esverb in its second mode, where the user manually searches for the root.

This command is required at the beginning of each session.

STEP 2

---- -

run mhdequil

This code sets up a cylindrical equilibrium for esverb based on user input. All data is entered in mks units. The code will prompt the user for:

awall
btheta/radius/ifit
pressure/radius/ifit
mass density/radius/ifit

The user should provide free-format inputs consisting of:

line 1: awall -- wall radius (meters)
line 2: btheta array -- input array of azimuthal magnetic fields (up to 11 values)
line 3: radius array -- input array giving radii at which magnetic fields were specified
line 4: ifit array -- input array of 0 or 1 for specifying whether linear (ifit=0) or 1/r (ifit=1) interpolations are to be used to specify the magnetic field for esverb.
line 5: pressure array -- pressure profile (nt/m**2)
line 6: radius array -- radii where pressure specified
line 7: ifit array -- linear,1/r fit switch for pressure
line 8: mass density -- density profile (kg/m**3)
line 9: radius array -- radii where density specified
line 10: ifit array -- linear,1/r switch for density

The code will generate input data for esverb, and store it in unformatted form on esverb.dat.

This step is needed only if esverb.dat does not already exist.

STEP 3

---- -

run read30

This code reads esverb.dat and allows the user to see the data he is feeding to esverb..

STEP 4

---- -

run esverb

This is the main code. It will read the equilibrium data from esverb.dat, and will prompt the user for additional information:

EVGUES,DEVAL,NEVAL,XK,XM,GAMMA,IBCL,IBCR,XL,XR.

These quantities are to be inputted in free-format as a single line input. They stand for the following data:

evgues -- initial guess for the eigenvalue, normalized as (omega*awall/vaa)**2, where omega**2 is the eigenvalue (a squared frequency),awall is the wall radius, and vaa is the Alfven speed at the wall.

deval -- eigenvalue increment for use when the code is used in mode 2 -- the code will examine the system for neval distinct choices of the eigenvalue, starting with evgues and incrementing the eigenvalue by deval to get each new choice.

neval -- the number of trial eigenvalues when the code is used in mode 2. If neval=0 is entered, the

code will run in the first mode, disregarding deval and using evgues as the first guess for the root finder.

xk -- the axial wavenumber, normalized as k*awall.

xm -- the azimuthal mode number.

samma -- the specific heat ratio, default value is 5/3

ibcl,ibcr -- boundary flags, default to dirichlet bc's

xl,xr -- grid limits, default to normalized 0,1 grid.

The code will also prompt (after some time) for a parameter, SETNRM, which allows the user to alter the normalization of the determinant. Typically, this parameter will be specified as 1, but if the determinant is close to either underflow or overflow on the VAX, specifying setnrm different from 1 may allow the calculation to proceed. Alternatively, the number of nodes carried by code can be reduced to avoid overflow/underflow.

The output from the code will be on egvprb.scr, and on plot files egvplt.dat (mode 1) or egcplt.dat (mode 2).

STEP 5

---- -

run egcplt (mode 2)

No user input is required. The code will generate plots of

1) det vs. ev -- the charcteristic determinant vs the eigenvalue.
2) a-coefficient vs. r for each eigenvalue (3-d plot)
3) b-coefficient vs. r for each eigenvalue (3-d plot)
4) c-coefficient vs. r for each eigenvalue (3-d plot)
5) eigenfunction vs. r for each eigenvalue

STEP 5'

---- --

run egvplt (mode 1)

User may specify which plots are desired. He specifies:

NGRAPHS -- # graphs to be generated

MGRAPH(J) -- switch for each type of graph
1,0 mean yes,no

mgraph(1) -- plot a-coefficient vs r
mgraph(2) -- plot b-coefficient vs r
mgraph(3) -- plot c-coefficient vs r
mgraph(4) -- plot eigenfunction vs r

*

*----------+

*

* END OF EGVPRB.INF

*

*----------+

*

$

APPENDIX E

Listing of EGVSETUP

```
$ASSIGN EGVPRB.SCR FOR003
$ASSIGN EGVPRB.DAT FOR030
$ASSIGN EGVPLT.DAT FOR015
$ASSIGN EGCOPLT.DAT FOR090
$
```

APPENDIX F

Listing of MHDEQUIL

```
      PROGRAM MHDEQUIL
C
C*****WRITES MHD EQUILIBRIUM DATA
C*****     FOR EGVPRB
C*****ON FILE FOR030.DAT
C
      parameter(np=21,
     *          np2p=np+2,
     *          ninp=11)
      dimension bth(np2p),press(np2p),rho(np2p),
     *            va(np2p),cs(np2p),physrd(np2p),
     *            finp(ninp),rinp(ninp),ifit(ninp)
      data awall/1./,xm0/1.256637e-6/
      data samma/1.66667/
      data xl,xr/0.,1./
      data ifit/ninp*0/
      rewind 30
      print 1000
1000  format(1x,'enter data in mks units'/
     *       3x,'awall'/
     *       3x,'btheta/radius/ifit'/
     *       3x,'pressure/radius/ifit'/
     *       3x,'mass density/radius/ifit'/
     *       10x,'ifit=(0,1)=(linear,1/r) fit to data')
      read(5,*) awall
      xr=awall
      del=xr-xl
      do 10 i=1,np
      fac=(float(i-1)/float(np-1))
10    physrd(i)=xl+del*fac
      read(5,*) finp
      read(5,*) rinp
      read(5,*) ifit
      if(rinp(1).ne.0.)     goto 500
      do 15 j=2,ninp
      if(rinp(j).eq.awall) rinp(j)=1.0001*awall
15    continue
      do 25 i=1,np
      r=physrd(i)
      do 20 j=1,ninp
      if(rinp(j).le.r) goto 20
      if(ifit(j).eq.0) bth(i)=finp(j-1)+(finp(j)-finp(j-1))
     * *(r-rinp(j-1))/(rinp(j)-rinp(j-1))
      if(ifit(j).eq.1) bth(i)=finp(j-1)*rinp(j-1)/r
      goto 25
20    continue
25    continue
c*****
      do 30 i=1,ninp
      ifit(i)=0
      rinp(i)=0.
30    finp(i)=0.
      read(5,*) finp
      read(5,*) rinp
      read(5,*) ifit
      if(rinp(1).ne.0.      goto 500
      do 32 j=1,ninp
      if(rinp(j).eq.awall) rinp(j)=1.0001*awall
32    continue
      do 40 i=1,np
      r=physrd(i)
      do 35 j=1,ninp
```

```
      if(rinp(j).le.r) goto 35
      if(ifit(j).eq.0) press(i)=finp(j-1)+(finp(j)-finp(j-1))
     * *(r-rinp(j-1))/(rinp(j)-rinp(j-1))
      if(ifit(j).eq.1) press(i)=finp(j-1)*rinp(j-1)/r
      goto 40
35    continue
40    continue
c*****
      do 45 i=1,ninp
      ifit(i)=0
      rinp(i)=0.
45    finp(i)=0.
      read(5,*) finp
      read(5,*) rinp
      read(5,*) ifit
      if(rinp(1).ne.0.)    goto 500
      do 47 j=2,ninp
      if(rinp(j).eq.awall) rinp(j)=1.0001*awall
47    continue
      do 55 i=1,np
      r=rhyrd(i)
      do 50 j=1,ninp
      if(rinp(j).le.r) goto 50
      if(ifit(j).eq.0) rho(i)=finp(j-1)+(finp(j)-finp(j-1))
     * *(r-rinp(j-1))/(rinp(j)-rinp(j-1))
      if(ifit(j).eq.1) rho(i)=finp(j-1)*rinp(j-1)/r
      goto 55
50    continue
55    continue
      write(6,1001)
1001  format(1h1,19x,'r',12x,'bth',12x,'rho',10x,'press',
     *       13x,'va',13x,'cs'/)
      do 60 i=1,np
      va(i)=sqrt(bth(i)**2/xm0/rho(i))
      cs(i)=sqrt(gamma*press(i)/rho(i))
      write(6,1002) rhyrd(i),bth(i),rho(i),press(i),va(i),cs(i)
1002  format(5x,6e15.5)
60    continue
      write(30) awall,bth,rho,press,va,cs
      endfile 30
      goto 600
500   write(6,1003) rinp(1)
1003  format(1h0,5x,'**********input error*****rinp(1)=0. is expected',
     *  '**********input has rinp(1)=',e15.5/)
600   continue
      stop
      end
$
```

APPENDIX G

Listings of READ15 and READ30

```
      program read15
      parameter(nsdp=101)
      dimension pltsrd(nsdp),pltr(nsdp),plti(nsdp)
      rewind 15
      do 100 m=1,4
      read(15) nsd,pltsrd,pltr,plti,ymin,ymax
      write(6,1000) nsd
1000  format(1h1,5x,'nsd=',i6)
      write(6,1001) (pltsrd(i), i=1,nsdp)
1001  format(5x,'pltsrd='/50(10x,6e15.5/))
      write(6,1002) (pltr(i), i=1,nsdp)
1002  format(5x,'pltr='/50(10x,6e15.5/))
      write(6,1003) (plti(i), i=1,nsdp)
1003  format(5x,'plti='/50(10x,6e15.5/))
      write(6,1004) ymin,ymax
1004  format(5x,'ymin=',e15.5,5x,'ymax=',e15.5)
100   continue
      stop
      end
$
```

```
      program read30
c*****************************************************
c     reads & prints for030 written by                *
c     by mhdequil to provide data to esvprb    *
c*****************************************************
      parameter(np=21,
     *          np2p=np+2)
      dimension bth(np2p),press(np2p),rho(np2p),
     *          va(np2p),cs(np2p),physrd(np2p)
      rewind 30
      read(30) awall,bth,rho,press,va,cs
      do 10 i=1,np
      fac=(float(i-1)/float(np-1))
10    physrd(i)=awall*fac
      write(6,1000) awall
1000  format(1h0,20x,'data for esvprb'/10x,'awall=',e15.5/)
      write(6,1001)
1001  format(5x,'i',9x,'r',7x,'bth',7x,'rho',5x,'press',
     *       8x,'va',8x,'cs')
      do 20 i=1,np
      write(6,1002) i,physrd(i),bth(i),rho(i),press(i),va(i),cs(i)
1002  format(1x,i5,6e10.3)
20    continue
      stop
      end
$
```

APPENDIX H

Listing of EGVPLT

```
      PROGRAM EGVPLT
C     IMPLICIT REAL*8(D)
      REAL IX1,IX2,IY1,IY2
      INTEGER OTAPE,BUFFER(1)
      LOGICAL LTIME
      PARAMETER(NGDP=101)
      DIMENSION DPLTGRD(NGDP),DPLTR(NGDP),DPLTI(NGDP),
     * PLTGRD(NGDP),PLTR(NGDP),PLTI(NGDP),MGRAPH(4)
      INTEGER TITLIN(1),TITEND(1)
      LENBUF=1
      OTAPE=106
      NGRAPHS=1
      X1R=150.
      X2R=750.
      Y1R=100.
      Y2R=700.
      NDVX=20
      NDVY=20
      LTIME=.FALSE.
      TIME=0.
      PRINT 900
900   FORMAT(1X,'NGRAPHS,MGRAPH'/
     * 5X,'MGRAPH(J)=(1,0)=(Y,N)'/
     * 5X,'    J=1,2,3,4=A,B,C,EV')
      READ(5,*) NGRAPHS,MGRAPH
      CALL GRAFIT(0,OTAPE,BUFFER,TITLIN)
      CALL GRAFIT(8,OTAPE,BUFFER,-1)
      REWIND 15
      NREAD=0
      DO 100 JGRAPH=1,NGRAPHS
5     NREAD=NREAD+1
      READ(15,END=150) NGD,DPLTGRD,DPLTR,DPLTI,DYMIN,DYMAX
      IF(MGRAPH(NREAD).EQ.0) GOTO 5
      YMIN=DYMIN
      YMAX=DYMAX
      YYMAX=MAX(ABS(YMAX),ABS(YMIN))
      IF(YYMAX.EQ.0.) YYMAX=1.
      YMAX=YMAX/YYMAX
      YMIN=YMIN/YYMAX
      DO 10 J=1,NGDP
      PLTGRD(J)=DPLTGRD(J)
      PLTR(J)=DPLTR(J)
10    PLTI(J)=DPLTI(J)
      DO 20 J=1,NGDP
      PLTR(J)=PLTR(J)/YYMAX
      PLTI(J)=PLTI(J)/YYMAX
20    CONTINUE
      X1=PLTGRD(1)
      X2=PLTGRD(NGD)
      IX1=X1
      IX2=X2
      Y1=YMIN
      Y2=YMAX
      IY1=Y1
      IY2=Y2
      CALL GRIGEN(X1,X2,X1R,X2R,Y1,Y2,Y1R,Y2R,
     * ' R $.',' LPMA $.',
     * TIME,IX1,IX2,IY1,IY2,NDVX,NDVY,LTIME)
      CALL PLOTL(PLTGRD,PLTR,NGD,'    $.')
      CALL PLOTL(PLTGRD,PLTI,NGD,' ...$.')
      CALL GRAFIT(4,OTAPE,BUFFER,JGRAPH)
100   CONTINUE
```

```
150     CALL GRAFIT(9,OTAPE,BUFFER,TITEND)
        STOP
        END
$
```

APPENDIX I

Listing of EGCOPLT

```
      PROGRAM EGCOPLT
      complex ev,det
      REAL IX1,IX2,IY1,IY2
      INTEGER OTAPE,BUFFER(1)
      LOGICAL LTIME
      PARAMETER(NGDP=101,
     *          nevalp=20)
      DIMENSION yymin(4),yymax(4),detv(nevalp),evv(nevalp),
     *          aa(4,nevalp,nsdp),xdata(nevalp,nsdp),ydata(nevalp,nsdp),
     *          xplot(nsdp),yplot(nsdp),
     * PLTGRD(NGDP),PLTR(NGDP),PLTI(NGDP),MGRAPH(4)
      INTEGER TITLIN(1),TITEND(1)
      data sphi/.866/,cphi/-.5/
      LENBUF=1
      OTAPE=106
      NGRAPHS=1
      X1R=150.
      X2R=750.
      Y1R=100.
      Y2R=700.
      NDVX=20
      NDVY=20
      LTIME=.FALSE.
      TIME=0.
      CALL GRAFIT(0,OTAPE,BUFFER,TITLIN)
      CALL GRAFIT(8,OTAPE,BUFFER,-1)
      REWIND 90
      read(90) neval
      do 8000 m=1,4
      yymin(m)=0.
 8000 yymax(m)=0.
      detmin=0.
      detmax=0.
      do 8010 i=1,neval
      read(90) ev,det
      evv(i)=real(ev)
      detv(i)=real(det)
      detmin=min(detmin,detv(i))
      detmax=max(detmax,detv(i))
      do 8008 M=1,4
      read(90) nsd,pltsrd,pltr,plti,ymin,ymax
      do 8005 j=1,nsd
 8005 aa(m,i,j)=pltr(j)
      yymin(m)=min(yymin(m),ymin)
 8008 yymax(m)=max(yymax(m),ymax)
 8010 continue
      do 8015 M=1,4
      ymaxx=max(abs(yymin(m)),abs(yymax(m)))
      yymin(m)=yymin(m)/ymaxx
      yymax(m)=yymax(m)/ymaxx
      do 8015 i=1,neval
      do 8015 j=1,nsd
 8015 aa(m,i,j)=aa(m,i,j)/ymaxx
C
      ddmax=max(abs(detmin),abs(detmax))
      do 20 i=1,neval
 20   detv(i)=detv(i)/ddmax
      detmax=detmax/ddmax
      detmin=detmin/ddmax
      evmin=min(evv(1),evv(neval))
      evmax=max(evv(1),evv(neval))
      evmaxx=max(abs(evmin),abs(evmax))
```

```
      do 8020 i=1,neval
8020  evv(i)=evv(i)/evmaxx
      evmax=evmax/evmaxx
      evmin=evmin/evmaxx
      evmin=min(evmin,0.)
      evmax=max(evmax,0.)
      detmin=min(detmin,0.)
      detmax=max(detmax,0.)
      X1=evmin
      X2=evmax
      IX1=X1
      IX2=X2
      Y1=detmin
      Y2=detmax
      IY1=Y1
      IY2=Y2
      jsraph=1
      CALL GRIGEN(X1,X2,X1R,X2R,Y1,Y2,Y1R,Y2R,
     * ' EV $.',' TED $.',
     * TIME,IX1,IX2,IY1,IY2,NDVX,NDVY,LTIME)
      CALL PLOTL(evv,detv,neval,'     $.')
      xplot(1)=evmin
      xplot(2)=evmax
      yplot(1)=0.
      yplot(2)=0.
      call plotl(xplot,yplot,2,' ...$.')
      CALL GRAFIT(4,OTAPE,BUFFER,JGRAPH)
100   CONTINUE
      do 8500 M=1,4
      ymin=0.
      ymax=0.
      xmin=0.
      xmax=0.
      do 8100 i=1,neval
      do 8050 j=1,nsd
      xdata(i,j)=pltsrd(j)+evv(i)*cphi
      ydata(i,j)=aa(m,i,j)-evv(i)*sphi
      xmin=min(xmin,xdata(i,j))
      xmax=max(xmax,xdata(i,j))
      ymin=min(ymin,ydata(i,j))
      ymax=max(ymax,ydata(i,j))
8050  continue
8100  continue
      xmaxx=max(abs(xmin),abs(xmax))
      ymaxx=max(abs(ymin),abs(ymax))
      do 8200 i=1,neval
      do 8200 j=1,nsd
      xdata(i,j)=xdata(i,j)/xmaxx
8200  ydata(i,j)=ydata(i,j)/ymaxx
      xmax=xmax/xmaxx
      xmin=xmin/xmaxx
      ymax=ymax/ymaxx
      ymin=ymin/ymaxx
      xmin=min(xmin,0.)
      xmax=max(xmax,0.)
      ymin=min(ymin,0.)
      ymax=max(ymax,0.)
      x1=xmin
      x2=xmax
      y1=ymin
      y2=ymax
      ix1=x1
      ix2=x2
```

```
      iv1=v1
      iv2=v2
      CALL GRIGEN(X1,X2,X1R,X2R,Y1,Y2,Y1R,Y2R,
     * ' R $.',' LPMA $.',
     * TIME,IX1,IX2,IY1,IY2,NDVX,NDVY,LTIME)
      do 8300 i=1,neval
      do 8250 j=1,nsd
      xplot(j)=xdata(i,j)
8250  yplot(j)=ydata(i,j)
      call plotl(xplot,yplot,nsd,'     $.')
8300  continue
      do 8400 i=1,neval
      xplot(1)=evv(i)*cphi/XMAXX
      xplot(2)=xplot(1)
      yplot(1)=(yymin(m)-evv(i)*sphi)/YMAXX
      yplot(2)=(yymax(m)-evv(i)*sphi)/YMAXX
      xplot(1)=max(xplot(1),xmin)
      xplot(2)=min(xplot(2),xmax)
      yplot(1)=max(yplot(1),ymin)
      yplot(2)=min(yplot(2),ymax)
      call plotl(xplot,yplot,2,'     $.')
      xplot(1)=evv(i)*cphi/XMAXX
      xplot(2)=(pltgrd(nsd)+evv(i)*cphi)/XMAXX
      yplot(1)=-evv(i)*sphi/YMAXX
      yplot(2)=-evv(i)*sphi/YMAXX
      xplot(1)=max(xplot(1),xmin)
      xplot(2)=min(xplot(2),xmax)
      yplot(1)=max(yplot(1),ymin)
      yplot(2)=min(yplot(2),ymax)
      call plotl(xplot,yplot,2,'     $.')
8400  continue
      xplot(1)=evmin*cphi/XMAXX
      xplot(2)=evmax*cphi/XMAXX
      yplot(1)=-evmin*sphi/YMAXX
      yplot(2)=-evmax*sphi/YMAXX
      call plotl(xplot,yplot,2,'     $.')
      CALL PLOTC(0.,0.,1,' ...$.')
C     XPLOT(1)=EVMIN*CPHI
C     YPLOT(1)=YYMAX(M)-EVMIN*SPHI
C     XPLOT(2)=EVMAX*CPHI
C     YPLOT(2)=YYMAX(M)-EVMAX*SPHI
C     XPLOT(3)=PLTGRD(NGD)+EVMAX*CPHI
C     YPLOT(3)=YPLOT(2)
C     XPLOT(4)=PLTGRD(NGD)+EVMIN*CPHI
C     YPLOT(4)=YPLOT(1)
C     XPLOT(5)=XPLOT(1)
C     YPLOT(5)=YPLOT(1)
C     XPLOT(6)=XPLOT(5)
C     YPLOT(6)=YYMIN(M)-EVMIN*SPHI
C     XPLOT(7)=XPLOT(4)
C     YPLOT(7)=YPLOT(6)
C     XPLOT(8)=XPLOT(3)
C     YPLOT(8)=YYMIN(M)-EVMAX*SPHI
C     XPLOT(9)=XPLOT(2)
C     YPLOT(9)=YPLOT(8)
C     XPLOT(10)=XPLOT(6)
C     YPLOT(10)=YPLOT(6)
C     XPLOT(11)=XPLOT(7)
C     YPLOT(11)=YPLOT(7)
C     XPLOT(12)=XPLOT(4)
C     YPLOT(12)=YPLOT(4)
C     XPLOT(13)=XPLOT(3)
C     YPLOT(13)=YPLOT(3)
```

```
C       XPLOT(14)=XPLOT(8)
C       YPLOT(14)=YPLOT(8)
C       XPLOT(15)=XPLOT(9)
C       YPLOT(15)=YPLOT(9)
C       XPLOT(16)=XPLOT(6)
C       YPLOT(16)=YPLOT(6)
C       DO 8900 II=1,16
C       IF(XPLOT(II).GT.XMAX) XPLOT(II)=XMAX
C       IF(XPLOT(II).LT.XMIN) XPLOT(II)=XMIN
C       IF(YPLOT(II).GT.YMAX) YPLOT(II)=YMAX
C       IF(YPLOT(II).LT.YMIN) YPLOT(II)=YMIN
8900    CONTINUE
C       CALL PLOTL(XPLOT,YPLOT,16,' ...$.')
        jsraph=m+1
        call srafit(4,otape,buffer,jsraph)
8500    continue
        call srafit(9,otape,buffer,titend)
        stop
        end
$
```

APPENDIX J

Listing of EGVPRB

```
c
c#################################################################
c
      subroutine bc(ibndry)
C     IMPLICIT REAL*8(A-H,O-Z)
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      common/baksub/aa(ndm2p,ndm2p)
      common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
      common/bdycds/bcl(3,neqp),bcr(3,neqp)
      common/matrix/  a(np2p,neqp,neqp),
     1                b(np2p,neqp,neqp),
     2                c(np2p,neqp,neqp)
      common/trsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
     1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
     2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
     3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
     4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
     6              crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
      common/spvals/phi(maxp),spvals(nsdp,np2p)
      common/grids/nsd,xl,xr,phygrd(np),pltgrd(nsdp)
      common/integer/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
c
      COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi
      COMPLEX a,b,c,det,detnrm
      COMPLEX bcl,bcr,znorm
c
c
c
      common/ncl/nm2,nm1,n,np1,np2
      common/rcl/r(np),rn(np)
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
```

```
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
c     setup boundary conditions
c
c     ibndry = 1    left  boundary condition
c     ibndry = 2    right boundary condition
c
      COMPLEX tembc(3),srftrm
      index = (ibndry-1)*np1*neq
c
      deli = 1./(xr-xl)
c
      do 90 i=1,neq
c
      go to (10,20) ibndry
c
   10 continue
      do 11 j=1,3
      tembc(j) = bcl(j,i)
   11 continue
      go to 30
   20 continue
      do 21 j=1,3
      tembc(j) = bcr(j,i)
   21 continue
   30 continue
c
      t1 = ABS(tembc(1))
      t2 = ABS(tembc(2))
      t3 = ABS(tembc(3))
c
      if(t1.ne.0..or.t2.ne.0.) go to 40
      print 100
      WRITE(3,100)
      stop  300
c
   40 if(t1.ne.0..or.t3.ne.0.) go to 41
      go to 90
c
   41 if(t2.ne.0..or.t3.ne.0.) go to 42
      do 50 k=1,ndm2
      aa(index+i,k) = (0.,0.)
   50 continue
      aa(index+i,index+i) = (1.,0.)
      go to 90
c
   42 if(t1.ne.0.) go to 43
      print 101
      WRITE(3,101)
      stop  301
c
   43 if(t2.ne.0.) go to 44
```

```
      print 102
      WRITE(3,102)
      stop  302
c
   44 if(t3.ne.0.) go to 45
      go to (61,62) ibndry
   61 continue
      srftrm = -deli*a(  1,i,i)*bc0**3*tembc(1)/tembc(2)
      go to 63
   62 continue
      srftrm =  deli*a(np2,i,i)*bc1**3*tembc(1)/tembc(2)
   63 continue
      aa(index+i,index+i) = aa(index+i,index+i) + srftrm
      go to 90
c
   45 continue
      print 104
      WRITE(3,104)
      stop  304
c
   90 continue
c
  100 format(1x,' stop 300  improper boundary conditions imposed')
  101 format(1x,' stop 301  boundary conditions not yet implemented')
  102 format(1x,' stop 302  boundary conditions not yet implemented')
  103 format(1x,' stop 303  boundary conditions not yet implemented')
  104 format(1x,' stop 304  boundary conditions not yet implemented')
c
      return
      end
      COMPLEX function cevalf(evtril,fcn)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. macmahon univ. of texas at austin  oct 1973
c*****modified by w. h. miner    univ. of texas at austin  jun 1976
c*****modified by a. a. mondelli science applications, inc. mar 1981
c********** Oct 26 -- 2.9
c********** oct 23 -- mate with fcn pks
c
      external fcn
c
      common/cevlf1/dx1,ftest,dftest,dxtst1,dxtst2,tstep,tquad
      common/cevlf2/imax,ieval,key,istep,jstart
      common/cevlf3/f1,x1
c
      COMPLEX dx1,xq(2),evtril,fcn
      COMPLEX x0,x1,x2
      COMPLEX f0,f1,f2
      COMPLEX a0,a1,a2
      COMPLEX a,b,c,d,df,x
c
      data dx1,ftest,dftest,dxtst1,dxtst2,tstep,tquad,imax/
     1     (.001,0.),1.e-08,1.e-09,1.e-08,1.,.2,.02,15/
c
c
c#########################################################################
c
c     dx1    initial step
c     ftest  tolerance on fcn
c     dftest minimum df (jexit=5)
c     dxtst1 allowed estimated error in ev
c     dxtst2 maximum dx (jexit=4)
c     tstep  defines 'small' step
c     tquad  test for neighboring root
```

```
c      imax  maximum number of calls to fcn
c
c#########################################################################
c
c      root solver modified to restrict search to REAL ROOTS ONLY
c      ---- ------ -------- -- -------- ------ -- ---- ----- ----
c
       data epsi2/ 1.e-16 /
       data jstart/0/
c
       DX1R=MIN(REAL(DX1),ABS(REAL(EVTRIL)/10.))
       TQUAD=DX1R**2
       DX1=CMPLX(DX1R,0.)
       DXTST1=DX1R**2/1000000.
       DXTST2=DX1R**2*100.
       DFTEST=FTEST/DX1R**2
c
       WRITE(3,1)
     1 format(/' i, evr,evi, abs(fcn), kode'/)
c
c         *********set up for first iteration
     2 istep=-1
       x2=evtril
       ieval=0
       key=0
       go to 10
c
c      *******test ieval, dx
     5 if(ieval.ge.imax)go to 99
       if(istep.eq.0)go  to  6
c
       d=x-x2
       t=REAL(d)**2 + AIMAG(d)**2
       if(t.lt.dxtst1) go to 110
       if(t.gt.dxtst2) go to 120
c      *******shift previous values
     6 f0=f1
       f1=f2
       x0=x1
       x1=x2
c       ****new x-values
       x2=x
c      *****new value of fcn
    10 ieval=ieval+1
       f2=fcn(x2)
c      ******** test for conversence
       af2s=REAL(f2)**2  + AIMAG(f2)**2
c
c
       WRITE(3,11)ieval,x2,af2s,istep
    11 format(1x,i3,2e13.5,5x,e10.2,1x,i4)
c
    14 if(af2s.lt.ftest) go to 100
c      ***** test for mode of next step
       if(istep)20,40,15
c
    15 if(t.lt.tstep) go to 40
c     ********* first step, or previous step large,  take new step dx1
    20 if(jstart.eq.1)go to 25
       x=x2+dx1
       istep=0
       go to 5
c
```

```
   25 jstart=0
      istep=0
c
c
c
c
   40 df=(f2-f1)/(x2-x1)
   41 t=REAL(df)**2 +AIMAG(df)**2
      if(t.lt.dftest) go to 130
c
   45 x=.5*(x1+x2-(f1+f2)/df)
      if(istep.ge.1)go to 50
      istep=istep+1
      go to 5
c
c     ******* quadratic extrapolation from points x0,x1,x2
   50 a0=f0/((x0-x1)*(x0-x2))
      a1=f1/((x1-x2)*(x1-x0))
      a2=f2/((x2-x0)*(x2-x1))
c
      a=a0+a1+a2
      istep=0
      t=REAL(a)**2 + AIMAG(a)**2
      if(t.lt.epsi2)go to 40
c
      b=a0*(x1+x2) +a1*(x2+x0) +a2*(x0+x1)
      b=-b
      c=a0*x1*x2 +a1*x2*x0 +a2*x0*x1
c
      d=SQRT(b**2-4.0*a*c)
c
      xq(1)=(-b+d)/(2.*a)
      xq(2)=(-b-d)/(2.*a)
      xqr1=real(xq(1))
      xqr2=real(xq(2))
      xq(1)=cmplx(xqr1,0.)
      xq(2)=cmplx(xqr2,0.)
c
      d=xq(1)-x
      t=REAL(d)**2 + AIMAG(d)**2
      d=xq(2)-x
      t1=REAL(d)**2 + AIMAG(d)**2
      i=1
      if(t1.gt.t)go to 55
      i=2
      t=t1
c
   55 df=2.*a*xq(i) +b
      x=xq(i)
      istep=1+i
      d=xq(1)-xq(2)
      t1=REAL(d)**2 + AIMAG(d)**2
      if(t1.gt.tquad)go to 60
c
      WRITE(6,66) xq(3-i)
      WRITE(3,66) XQ(3-I)
      istep=3+i
c
   60 if(t.lt.dxtst1) go to 110
      go to 5
c
c
   66 format(1x,'warning, neighboring root at ',2e13.5/)
```

```
c
c       ****** ieval . gt. imax-----exit
   99 key=4
      jexit=6
c       ****** conversence
  100 cevalf=x2
      write(6,112) ieval,x2,istep,key,jexit
      write(3,112) ieval,x2,istep,key,jexit
  112 format(1x,i3,'   conversed root=',2e13.5,
     .       '   istep,key,jexit= ',3i5/)
      return
c       ****** problems
  110 key=1
      if(istep.lt.1)go to 200
      cevalf=x
      WRITE(6,111)ieval,x,istep
      WRITE(3,111) IEVAL,X,ISTEP
      return
c
  111 format(1x,i3,2e13.5,' del x .lt. dxtst1',i4)
c
  120 key=2
      go to 200
  130 key=3
      go to 200
  140 key=5
      go to 200
  200 WRITE(6,201)key
      WRITE(3,201) KEY
      jexit=key+2
      if(key.eq.5) go to 5
      go to 100
  201 format(1x,' problem key=',i3)
      end
c
      COMPLEX function fcn(ev)
C     IMPLICIT REAL*8(A-H,O-Z)
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      common/baksub/aa(ndm2p,ndm2p)
      common/eqnvcs/eqvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
      common/bdycds/bcl(3,neqp),bcr(3,neqp)
      common/matrix/  a(np2p,neqp,neqp),
     1                b(np2p,neqp,neqp),
     2                c(np2p,neqp,neqp)
      common/tnsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
     1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
     2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
     3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
     4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
```

```
      6               crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
       common/spvals/phi(maxp),spvals(nsdp,np2p)
       common/grids/nsd,xl,xr,physrd(np),pltsrd(nsdp)
       common/intser/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
       COMMON/ESTORE/EVSTORE
c
       COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi
       COMPLEX a,b,c,det,detnrm
       COMPLEX bc1,bcr,znorm,EVSTORE
c
c
c
       common/nc1/nm2,nm1,n,np1,np2
       common/rc1/r(np),rn(np)
       common/sac1/sa(nsap)
       common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
       common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
       common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
       common/phii1/pi1(np)/phii2/pi2(np)
       common/phii3/pi3(np)/phii4/pi4(np)
       common/phi4/e(np2p)/phi5/f(np2p)
       common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
       common/errc1/err(np2p)
       common/serrc1/amxer1,amxer2,ermax
       common/dum1/d1(np2p)/dum2/d2(np2p)
c
c      commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
       common/ssic1/ssi(np2p,nsip)
       common/dsic1/dsi(np2p,7,ndip)
       common/tsic1/tsi(np2p,49,ntip)
c
       common/nssip/nsi,nsj(nsip),nsv(2,nsip)
       common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
       common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
       equivalence (sa,sa3)
       dimension sa3(4,4,np2p)
c
       common/logic1/lintal,ldtnrm,lrerdr
       logical lintal,ldtnrm,lrerdr
c
       dimension prntl(4)
       data prntl/' coe','ff m','atri','x   '/
       COMPLEX temp
c
       common/time/t
       data immax/1/
c
       if(.not.lintal) go to 31
c
c---initialize values.
c
       T=GETIME(dum)
c
       call matrix(ev)
c
       T=GETIME(dum)-t
```

```
      print 904,t
      WRITE(3,904) T
c
      del=xr-xl
      deli=1./del
      del2i=deli*deli
      amxerr=.05
      amxer2=.4
c
c
      print 100,xl,xr
      WRITE(3,100) XL,XR
  100 format(1x,'values initialized  xl=',e12.5,'  xr=',e12.5)
c
c---chose grid.
      call grid(r,1)
c
c------ setup grid for plots ------------
c
      do 50 i=1,nsd
      pltgrd(i)=(i-1.0)/(nsd-1.0)
   50 continue
      print 102
      WRITE(3,102)
  102 format(1x,'grid chosen')
      print 101,(r(i),i=1,n)
      WRITE(3,101) (R(I),I=1,N)
  101 format(1x,e15.7)
c
c---set up splines.
      call bsplcf
c
      do 950 i=1,4
      do 950 j=1,4
      WRITE(3,910) (sa3(i,j,k),k=1,np2)
  950 continue
c
      T=GETIME(dum)-t
      print 901,t
      WRITE(3,901) T
c
c---compute intesrals.
      call opset
c
      T=GETIME(dum)-t
      print 902,t
      WRITE(3,902) T
c
c---set up decomposer.
      call deset(0)
c
      WRITE(3,198)
      WRITE(3,199) bc0,bc0p,bc0p1,bc1p1,bc1p,bc1
  198 format(1x,' spline boundary values')
c
      T=GETIME(dum)-t
      print 903,t
      WRITE(3,903) T
c
   31 continue
c
      temp=(1.,0.)
c
```

```
c----- initialize eisenvectors ------------
c
      do 92 i=1,ndm2
      usi(i) =(1.,0.)
   92 continue
c
      print 105
      WRITE(3,105)
  105 format(1x,'initial eisenvectors')
      WRITE(3,104) (USI(I),I=1,NDM2)
   94 continue
c
  104 format(1x,2e12.5)
c
c
c------initialize error vector--------
      do 93 i=1,n
      err(i)=0.
   93 continue
c
      WRITE(3,106)
  106 format(1x,'error vector')
      WRITE(3,101) (err(i),i=1,n)
c
c----- calculate matrix elements ----------
c
      im=1
   83 continue
c
c
c---------- store physical mesh ----------
c
      do 42 i=1,n
      physrd(i)=r(i)*del+xl
   42 continue
c
c
      call matrix(ev)
c
      T=GETIME(dum)-t
      print 904,t
      WRITE(3,904) T
c
      do 11 i=1,n
      do 11 j=1,neq
      do 11 k=1,neq
      ar(i,j,k)= REAL(a(i,j,k))
      ai(i,j,k)=AIMAG(a(i,j,k))
      br(i,j,k)= REAL(b(i,j,k))
      bi(i,j,k)=AIMAG(b(i,j,k))
      cr(i,j,k)= REAL(c(i,j,k))
      ci(i,j,k)=AIMAG(c(i,j,k))
c
   11 continue
c
c
c----- setup matrix elements in splines
c
      do 22 i=1,neq
      do 21 k=1,neq
      call dense(ar(1,i,k),ars(1,i,k))
      call dense(ai(1,i,k),ais(1,i,k))
      call dense(br(1,i,k),brs(1,i,k))
```

```
      call dersе(bi(1,i,k),bis(1,i,k))
      call dersе(cr(1,i,k),crs(1,i,k))
      call dersе(ci(1,i,k),cis(1,i,k))
   21 continue
   22 continue
c
      do 190 k=1,neq
      do 190 j=1,neq
      do 191 i=1,n
      WRITE(3,199) ar(i,j,k),ai(i,j,k),
     1           br(i,j,k),bi(i,j,k),
     1           cr(i,j,k),ci(i,j,k)
  191 continue
c
      do 192 i=1,np2
      WRITE(3,199) ars(i,j,k),ais(i,j,k),
     1           brs(i,j,k),bis(i,j,k),
     1           crs(i,j,k),cis(i,j,k)
  192 continue
  190 continue
  199 format(1x,6e12.5)
c
c
      T=GETIME(dum)-t
      print 905,t
      WRITE(3,905) T
c
      if(.not.lintal) go to 81
c
      if(.not.lrerdr) go to 85
c
c-----calculate new grid if necessary---------
c
      if(im.ge.immax) go to 85
c
      do 82 i=1,neq
      call splerr(ars(1,i,i),err)
      call splerr(ais(1,i,i),err)
      call splerr(brs(1,i,i),err)
      call splerr(bis(1,i,i),err)
      call splerr(crs(1,i,i),err)
      call splerr(cis(1,i,i),err)
   82 continue
      WRITE(3,6000) (err(mm),mm=1,n)
      igrid=2
      if(im.ne.0) igrid=3
      call grid(err,igrid)
      call rmove(err)
      call bsplcf
      call deset(0)
      im=im+1
      WRITE(3,6000) (r(mm),mm=1,n)
 6000 format(1x,10e12.3)
      if(im.lt.immax) go to 83
c
c------end of knot adjustment------------
c
   85 continue
c
c-----calculate intesrals----------
c
      call sfeval
c
```

```
      T=GETIME(dum)-t
      print 906,t
      WRITE(3,906) T
C
C
C     setup spline values for reconstruction
      do 41 j=1,np2
      call spvl(j,pltsrd(1),spvals(1,j),nsd)
   41 continue
C
      lintal=.false.
C
   81 continue
C
C-----------------------------------------------
C
      do 27 k=1,neq
      do 27 j=1,neq
      do 27 i=1,np2
      a(i,j,k)=DCMPLX(ars(i,j,k),ais(i,j,k))
      b(i,j,k)=DCMPLX(brs(i,j,k),bis(i,j,k))
      c(i,j,k)=DCMPLX(crs(i,j,k),cis(i,j,k))
   27 continue
C
      call setbc(ev)
C
C
C     zero coefficient matrix before each iteration
C
      do 70 i=1,ndm2
      do 70 j=1,ndm2
      aa(i,j) = (0.,0.)
   70 continue
C
C
      do 23 k=1,np2
C
      imin=max0(k-3,1)
      imax=min0(k+3,np2)
      do 26 i=imin,imax
      i1=i-k+4
      jmin=max0(1,k-3)
      jmax=min0(np2,k+3)
      do 20 l=1,neq
      do 20 m=1,neq
      do 24 j=jmin,jmax
      j1=j-k+4
      i2=i1+7*(j1-1)
```

```
      indr = (k-1)*neq + l
      indc = (i-1)*neq + m
      aa(indr,indc) = aa(indr,indc)
     1                        - del2i*a(j,l,m)*(tsi(k,i2,1)+tsi(k,i2,2))
     2                        +  deli*b(j,l,m)* tsi(k,i2,3)
     3                        +       c(j,l,m)* tsi(k,i2,4)
   24 continue
   20 continue
   26 continue
c
c     setup boundary conditions
c
      if(k.ne.1.and.k.ne.nf2) go to 71
      if(k.ne.1) go to 72
      call bc(1)
      go to 71
   72 continue
      call bc(2)
   71 continue
   23 continue
c
      if(.not.ldtnrm) call setnrm
c
      if(ldtnrm) go to 970
C     call print(printl,aa,2*ndm2,ndm2,1,2*ndm2,1,ndm2)
  970 continue
c
      call matnrm
c
      call leqt1c(aa,ndm2,ndm2,esvs,1,ndm2,1,wa,det,ier)
c
      if(ldtnrm) go to 980
C     call print(printl,aa,2*ndm2,ndm2,1,2*ndm2,1,ndm2)
  980 continue
c
      temp = det
c
c
c      normalization of determinant
c
      if(.not.ldtnrm) go to 90
      temp=temp/detnrm
      go to 91
   90 continue
      detnrm=temp
      ldtnrm=.true.
   91 continue
      nevss=2
      print 700,ev,temp
      WRITE(3,700) EV,TEMP
  700 format(1x,'ev guess=',2(e15.8,2x),'det=',2(e15.8,2x))
      fcn=temp
      EVSTORE=EV
  900 format(1x,' call to grid   complete ',1pe12.5)
  901 format(1x,' call to bsplcf complete ',1pe12.5)
  902 format(1x,' call to gpset  complete ',1pe12.5)
  903 format(1x,' call to deset  complete ',1pe12.5)
  904 format(1x,' call to matrix complete ',1pe12.5)
  905 format(1x,' call to defse  complete ',1pe12.5)
  906 format(1x,' call to sfeval complete ',1pe12.5)
  910 format(1x,13e10.2)
      return
      end
```

```
      subroutine matnrm
C     IMPLICIT REAL*8(A-H,O-Z)
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      common/baksub/aa(ndm2p,ndm2p)
      common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
      common/bdycds/bcl(3,neqp),bcr(3,neqp)
      common/matrix/  a(np2p,neqp,neqp),
     1                b(np2p,neqp,neqp),
     2                c(np2p,neqp,neqp)
      common/tnsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
     1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
     2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
     3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
     4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
     6              crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
      common/spvals/phi(maxp),spvals(nsdp,np2p)
      common/grids/nsd,xl,xr,physrd(np),pltsrd(nsdp)
      common/intser/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
c
      COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi
      COMPLEX a,b,c,det,detnrm
      COMPLEX bcl,bcr,znorm
c
c
c
      do 10 i=1,ndm2
      do 10 j=1,ndm2
      aa(j,i) = aa(j,i)*anormi
   10 continue
c
      return
c
      entry setnrm
      anorm = ABS(aa(5,5))
      PRINT 9000
      WRITE(3,9000)
9000  FORMAT(1X,'ENTER NORMALIZATION FACTOR - SETNRM')
      READ(5,*) FACNORM
      WRITE(6,*) FACNORM
      WRITE(3,*) FACNORM
      ANORM=ANORM*FACNORM
      anormi = 1./anorm
      return
      end
      subroutine matrix(ev)
C     IMPLICIT REAL*8(A-H,O-Z)
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
```

```
      3           nsip=  1,
      4           ndip=  1,
      5           ntip=  4,
      6           nsdp=101,
      7           nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
      7           neqsqp=neqp*neqp,nep=4*neqp+1,
      3           maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
      4           ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
      5           ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
       DIMENSION BTH(NP2P),PRESS(NP2P),RHO(NP2P),
     * XNU(NP2P),VA(NP2P),CS(NP2P),AAAA(NP2P),CCCC(NP2P),
     * ALPHA(NP2P),QQ(NP2P),ALPHP(NP2P),
     * ALPHR(NP2P),ALPHI(NP2P),ALPHRS(NP2P),ALPHIS(NP2P),
     * ALPHRP(NP2P),ALPHIP(NP2P)
       DIMENSION FFF(NP2P),FFR(NP2P),FFI(NP2P),FFRS(NP2P),FFIS(NP2P),
     *           FFRP(NP2P),FFIP(NP2P),FFP(NP2P)
       DIMENSION GGG(NP2P),GGSPL(NP2P),GGP(NP2P)
       common/baksub/aa(ndm2p,ndm2p)
       common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
       common/bdycds/bcl(3,neqp),bcr(3,neqp)
       common/matrix/  a(np2p,neqp,neqp),
     1                 b(np2p,neqp,neqp),
     2                 c(np2p,neqp,neqp)
       common/tnsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
     1                br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
     2                cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
     3               ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
     4               brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
     6               crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
       common/spvals/phi(maxp),spvals(nsdp,np2p)
       common/grids/nsd,xl,xr,phygrd(np),pltgrd(nsdp)
       common/intser/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
c
       COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi
       COMPLEX a,b,c,det,detnrm,deval
       COMPLEX bcl,bcr,znorm
       COMPLEX AAA,CCC,AAAA,CCCC,FFF,ALPHA,QQ,FFP,ALPHP
c
c
c
       common/nc1/nm2,nm1,n,np1,np2
       common/rc1/r(np),rn(np)
       common/sac1/sa(nsap)
       common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
       common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
       common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
       common/phii1/pi1(np)/phii2/pi2(np)
       common/phii3/pi3(np)/phii4/pi4(np)
       common/phi4/e(np2p)/phi5/f(np2p)
       common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
       common/errc1/err(np2p)
       common/serrc1/amxer1,amxer2,ermax
       common/dum1/d1(np2p)/dum2/d2(np2p)
       common/coef1/deval,neval
c
c      commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
```

```
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
      logical lintl
      dimension ibcl(neqp),ibcr(neqp)
      COMPLEX evsues,ca
      data lintl/.true./
      data xl,xr/0.,1./
      DATA GAMMA/1.66667/,ITRMAX/1/
      data ibcl,ibcr/neqp*1,neqp*1/
      DATA AWALL/1./,XMO/1.256637E-6/
c
      COMPLEX q,qp,argq,qsqrt
c
c     the choices of boundary conditions are:
c                 = 0  derivative = 0
c     ibcl,ibcr   = 1  function   = 0
c                 = 2  w. k. b.
c
c
      if (lintl) go to 1
C*****CALCULATION OF COEFFICIENTS
      PHYGRD(1)=1.E-5
      VA(1)=VA(2)*PHYGRD(1)/PHYGRD(2)
      DO 210 I=1,N
      RR=PHYGRD(I)
      CS2=CS(I)**2
      VA2=VA(I)**2
      AAA=VA2*(XM/RR)**2-EV
      CCC=CS2*AAA-EV*VA2
      AAAA(I)=AAA
      CCCC(I)=CCC
C*****LAMBDA11/LAMBDA12 CALCULATION
      FFF(I)=-RHO(I)/RR*(CCC+2.*EV*CS2)*AAA
     *       /(EV*EV+CCC*(XK*XK+(XM/RR)**2))
C*****(P*)'/R CALCULATION
      GGG(I)=RHO(I)*VA2/RR**2
C*****CALCULATION OF ALPHA
      ALPHA(I)=RHO(I)*RR*AAA*CCC/(EV*EV
     *          +CCC*(XK*XK+(XM/RR)**2))
210   CONTINUE
      DO 215 I=1,N
      FFR(I)=REAL(FFF(I))
      FFI(I)=AIMAG(FFF(I))
      ALPHR(I)=REAL(ALPHA(I))
      ALPHI(I)=AIMAG(ALPHA(I))
215   CONTINUE
C*****
      CALL DEPSE(FFR(1),FFRS(1))
      CALL REPSP(FFRS(1),FFRP(1))
      CALL DEPSE(FFI(1),FFIS(1))
      CALL REPSP(FFIS(1),FFIP(1))
C*****
      CALL DEPSE(GGG(1),GGSPL(1))
```

```
      CALL REPSP(GGSPL(1),GGP(1))
C*****
      CALL DEPSE(ALPHR(1),ALPHRS(1))
      CALL REPSP(ALPHRS(1),ALPHRP(1))
      CALL DEPSE(ALPHI(1),ALPHIS(1))
      CALL REPSP(ALPHIS(1),ALPHIP(1))
C*****
      DO 255 I=1,N
      FFP(I)=CMPLX(FFRP(I),FFIP(I))*DELI
      GGP(I)=GGP(I)*DELI
255   ALPHP(I)=CMPLX(ALPHRP(I),ALPHIP(I))*DELI
C*****
      DO 220 I=1,N
      RR=PHYGRD(I)
      CS2=CS(I)**2
      VA2=VA(I)**2
      AAA=AAAA(I)
      CCC=CCCC(I)
      QQ(I)=RR*(FFF(I)*2./RR*(1.+EV*CS2/CCC)
     *      +RR*GGP(I)+RHO(I)*(4.*VA2*(1.
     *      +EV*CS2/CCC)/RR**2-AAA)+FFP(I))
220   CONTINUE
C*****
      do 2 i=1,n
      x = phygrd(i)
      a(i,1,1) = ALPHA(I)
      b(i,1,1) = ALPHP(I)
      c(i,1,1) = QQ(I)
    2 continue
      return
c
c
      entry setbc(ev)
c
      do 90 i=1,neq
c
      ibranch = ibcl(i) + 1
c
      go to (10,20,30) ibranch
   10 continue
      bcl(1,i) =       (0.,0.)
      bcl(2,i) =       (1.,0.)
      bcl(3,i) =       (0.,0.)
      go to 90
   20 continue
      bcl(1,i) =       (1.,0.)
      bcl(2,i) =       (0.,0.)
      bcl(3,i) =       (0.,0.)
      go to 90
   30 continue
      q  =  c(1,i,i)*bc0
      qp = (c(1,i,i)*bc0p+c(2,i,i)*bc0p1)*deli
      arsq = q
      if(REAL(arsq).le.0..and.AIMAG(arsq).gt.0.) arsq=conjg(arsq)
      qsqrt = SQRT(arsq)
      if(REAL(q).lt.0..and.AIMAG(q).eq.0.) qsqrt = -qsqrt
      bcl(1,i) =-.25*qp/q+(0.,1.)*qsqrt
      bcl(2,i) = DCMPLX(1.,0.)
      bcl(3,i) =       (0.,0.)
c
   90 continue
c
      do 92 i=1,neq
```

```
c
      ibrnch = ibcr(i) + 1
c
      go to (12,22,32) ibrnch
   12 continue
      bcr(1,i) =       (0.,0.)
      bcr(2,i) =       (1.,0.)
      bcr(3,i) =       (0.,0.)
      go to 92
   22 continue
      bcr(1,i) =       (1.,0.)
      bcr(2,i) =       (0.,0.)
      bcr(3,i) =       (0.,0.)
      go to 92
   32 continue
      q  =  c(np2,i,i)*bc1
      qp = (c(np2,i,i)*bc1p+c(np1,i,i)*bc1p1)*deli
      argq = q
      if(REAL(argq).le.0..and.AIMAG(argq).gt.0.) argq=conjg(argq)
      qsqrt = SQRT(argq)
      if(REAL(q).lt.0..and.AIMAG(q).eq.0.) qsqrt = -qsqrt
      bcr(1,i) =-.25*qp/q-(0.,1.)*qsqrt
      bcr(2,i) = DCMPLX(1.,0.)
      bcr(3,i) =       (0.,0.)
c
   92 continue
      return
    1 continue
      WRITE(6,1000)
      WRITE(3,1000)
      read(5,*) evgues,deval,neval,xk,xm,gamma,ibcl,ibcr,xl,xr,
     .          itrmax
      write(6,*) evgues,deval,neval,xk,xm,gamma,ibcl,ibcr,xl,xr,
     .           itrmax
      write(3,*) evgues,deval,neval,xk,xm,gamma,ibcl,ibcr,xl,xr,
     .           itrmax
      ev = evgues
      REWIND 30
      READ(30) AWALL,BTH,RHO,PRESS,VA,CS
C*****NORMALIZATION
      VAA=VA(N)
      RHON=RHO(1)
      WRITE(6,1001)
      WRITE(3,1001)
      DO 800 I=1,N
      VA(I)=VA(I)/VAA
      CS(I)=CS(I)/VAA
      RHO(I)=RHO(I)/RHON
      WRITE(6,1002)I,VA(I),CS(I),RHO(I)
      WRITE(3,1002)I,VA(I),CS(I),RHO(I)
800   CONTINUE
C*****
      deli = 1./(xr-xl)
      lintl=.false.
 1000 format(1x,'enter namelist data'/
     * 3X,'EVGUES,DEVAL,NEVAL,XK,XM,GAMMA,IBCL,IBCR,XL,XR,',
     * 'itrmax'/
     * 3X,'EVGUES=(OMEGA*AWALL/VAA)**2,XK=K*AWALL')
1001  FORMAT(3X,'NORMALIZED EQUILIBRIUM PARAMETERS'/
     * 8X,'I',10X,'VA(I)',10X,'CS(I)',9X,'RHO(I)')
1002  FORMAT(3X,I5,3E15.5)
      return
      end
```

```
      subroutine nrmfcn
C     IMPLICIT REAL*8(A-H,O-Z)
c
c--------eisenfunctions normalized such that the value of the
c       central eisenfunction is (1.,0.) at x=0.
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      common/baksub/aa(ndm2p,ndm2p)
      common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
      common/bdycds/bcl(3,neqp),bcr(3,neqp)
      common/matrix/  a(np2p,neqp,neqp),
     1                b(np2p,neqp,neqp),
     2                c(np2p,neqp,neqp)
      common/tnsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
     1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
     2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
     3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
     4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
     6              crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
      common/spvals/phi(maxp),spvals(nsdp,np2p)
      common/grids/nsd,xl,xr,phygrd(np),pltgrd(nsdp)
      common/intger/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
      common/norm/znorm
c
      COMPLEX ev,evold,esvs,usi,cmxe,aa,al,phi
      COMPLEX a,b,c,det,detnrm
      COMPLEX bcl,bcr,znorm
c
c
c
      common/ncl/nm2,nm1,n,np1,np2
      common/rcl/r(np),rn(np)
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline intesrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single intesrals
c#### j=number of double intesrals
c#### k=number of triple intesrals
```

```
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      do 60 iter=1,itrmax
c
      call leqtlc(aa,ndm2,ndm2,usi,1,ndm2,2,wa,det,ier)
c
c
c
      do 50 i=1,neq
      do 50 j=1,np2
      esvs(j,i) = usi(i+(j-1)*neq)
   50 continue
c
c----- diagnostic print ----
c
      do 40 i=1,neq
      do 40 j=1,np2
      WRITE(3,100) esvs(j,i)
   40 continue
   60 continue
c
c
  100 format(1x,2e12.5)
  101 format(1x,13e10.2)
c
c
c----- setup eigenfunctions -------
c
    2 continue
      do 32 i=1,max
      phi(i)=(0.,0.)
   32 continue
      do 22 k=1,neq
      do 22 i=1,nsd
      index=(k-1)*nsd+i
      do 22 j=1,np2
      phi(index)=phi(index)+esvs(j,k)*spvals(i,j)
   22 continue
c
c     normalize eigenfunctions
c
      znorm=phi(1)
      do 24 i=2,max
      if(ABS(phi(i)).gt.ABS(znorm)) znorm=phi(i)
   24 continue
      IF(ABS(ZNORM).LT.1.E-20) ZNORM=(1.,0.)
      znorm=cmplx(abs(znorm),0.)
c
      return
      end
c
c-----------------------------------------------------
```

```
c
      subroutine offset
C     IMPLICIT REAL*8(A-H,O-Z)
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline intesrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single intesrals
c#### j=number of double intesrals
c#### k=number of triple intesrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      data nsj/nsip*0/
      data ndj/ndip*0/
      data ntj/ntip*0/
      data nsv/0,0/
      data ndv/1,0,1,0/
      data ntv/0,0,1,0,1,0,
     2         1,0,1,0,0,0,
     3         0,0,1,0,0,0,
     4         0,0,0,0,0,0/
      return
      end
      subroutine fltfcn
```

```
C      IMPLICIT REAL*8(A-H,O-Z)
c
       PARAMETER(NP   = 21,
      2          neqp=  1,
      3          nsip=  1,
      4          ndip=  1,
      5          ntip=  4,
      6          nsdp=101,
      7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
      7          neqsqp=neqp*neqp,nep=4*neqp+1,
      3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
      4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
      5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
       common/baksub/aa(ndm2p,ndm2p)
       common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
       common/bdycds/bcl(3,neqp),bcr(3,neqp)
       common/matrix/  a(np2p,neqp,neqp),
      1                b(np2p,neqp,neqp),
      2                c(np2p,neqp,neqp)
       common/trsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
      1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
      2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
      3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
      4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
      6              crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
       common/spvals/phi(maxp),spvals(nsdp,np2p)
       common/grids/nsd,xl,xr,phygrd(np),pltgrd(nsdp)
       common/intger/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
       common/norm/znorm
c
       COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi
       COMPLEX a,b,c,det,detnrm
       COMPLEX bcl,bcr,znorm
c
c
       dimension f(nsdp),pltr(nsdp),plti(nsdp)
c
       common/ncl/nm2,nm1,n,np1,np2
c
       COMPLEX temp(np2p,neqp,neqp)
c
c      REWIND 15
       do 2 i=1,nsd
     2 pltgrd(i)=(xr-xl)*pltgrd(i)+xl
       do 100 m=1,4
       DO 3 I=1,NGD
     3 PHI(I)=(0.,0.)
       go to(10,20,30,40)m
    10 continue
       do 11 i=1,neq
       do 11 j=1,neq
       do 11 k=1,np2
       temp(k,j,i) = a(k,j,i)
    11 continue
       go to 50
    20 continue
       do 21 i=1,neq
       do 21 j=1,neq
       do 21 k=1,np2
       temp(k,j,i) = b(k,j,i)
    21 continue
```

```
      go to 50
   30 continue
      do 31 i=1,neq
      do 31 j=1,neq
      do 31 k=1,np2
      temp(k,j,i) = c(k,j,i)
   31 continue
      go to 50
   40 continue
      do 41 j=1,neq
      do 41 k=1,np2
      temp(k,1,j) = esvs(k,j)/znorm
   41 continue
   50 continue
c
      do 60 i=1,neq
c
      do 61 j=1,neq
      do 62 k=1,nsd
      do 62 l=1,np2
      phi(k) = phi(k)+temp(l,i,j)*spvals(k,l)
   62 continue
c
      do 64 mm=1,nsd
      pltr(mm) =  REAL(phi(mm))
      plti(mm) = AIMAG(phi(mm))
   64 continue
c
      call extrma(pltr,plti,nsd,ymin,ymax)
      xstp = .1*(xr-xl)
      ystp = .2*(ymax-ymin)
      WRITE(90) NGD,PLTGRD,PLTR,PLTI,YMIN,YMAX
C     call title('program esvlet$',-100,'domain',6,
C    2           'amplitude',9,9.,6.)
C     call graf(pltsrd(1),xstp,pltsrd(nsd),ymin,ystp,ymax)
C     call curve(pltsrd,pltr,nsd,0)
C     call curve(pltsrd,plti,nsd,0)
C     call frame
c
      do 68 nn=i,nsd
   68 phi(nn) = (0.,0.)
c
   61 continue
      if(m.eq.4) goto 101
   60 continue
  100 continue
```

```
101    continue
       return
       end
       subroutine extrma(t1,t2,n,x1,x2)
C      IMPLICIT REAL*8(A-H,O-Z)
       dimension t1(1),t2(1)
       x1=t1(1)
       x2=t1(1)
       do 1 i=1,n
       if(x1.gt.t1(i)) x1=t1(i)
       if(x1.gt.t2(i)) x1=t2(i)
       if(x2.lt.t1(i)) x2=t1(i)
       if(x2.lt.t2(i)) x2=t2(i)
1      continue
       return
       end
       program sysode
C      IMPLICIT REAL*8(A-H,O-Z)
c
c
       PARAMETER(NP   = 21,
      2          neqp=  1,
      3          nsip=  1,
      4          ndip=  1,
      5          ntip=  4,
      6          nsdp=101,
      7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
      7          neqsqp=neqp*neqp,nep=4*neqp+1,
      3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
      4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
      5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
       common/losicl/lintal,ldtnrm,lrerdr
       logical lintal,ldtnrm,lrerdr
c
       common/baksub/aa(ndm2p,ndm2p)
       common/esnvcs/esvs(np2p,neqp),usi(ndm2p),wa(ndm2p)
       common/bdycds/bcl(3,neqp),bcr(3,neqp)
       common/matrix/  a(np2p,neqp,neqp),
      1                b(np2p,neqp,neqp),
      2                c(np2p,neqp,neqp)
       common/tnsfrm/ ar(np2p,neqp,neqp), ai(np2p,neqp,neqp),
      1               br(np2p,neqp,neqp), bi(np2p,neqp,neqp),
      2               cr(np2p,neqp,neqp), ci(np2p,neqp,neqp),
      3              ars(np2p,neqp,neqp),ais(np2p,neqp,neqp),
      4              brs(np2p,neqp,neqp),bis(np2p,neqp,neqp),
      6              crs(np2p,neqp,neqp),cis(np2p,neqp,neqp)
       common/spvals/phi(maxp),spvals(nsdp,np2p)
       common/grids/nsd,xl,xr,physrd(np),pltsrd(nsdp)
       common/intger/neq,neqsq,ndm2,ndm3,ndm23,ndm33,max,itrmax
c
       COMPLEX ev,evold,esvs,usi,cmxe,aa,a1,phi,ELF
       COMPLEX a,b,c,det,detnrm,deval
       COMPLEX bcl,bcr,znorm
c
c
c
       common/ncl/nm2,nm1,n,np1,np2
       common/rcl/r(np),rn(np)
       common/sacl/sa(nsap)
       common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
       common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
       common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
```

```
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
      common/coef1/deval,neval
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
      COMMON/ESTORE/EVSTORE
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      data nm2,nm1,n,np1,np2/nm2p,nm1p,np,np1p,np2p/
      data nsi,ndi,nti/nsip,ndip,ntip/
      data neq,nsd,neqsq,ne,max,nsa/neqp,nsdp,neqsqp,nep,maxp,nsap/
      data na1,ndm2,ndm3,ndm23,ndm33/na1p,ndm2p,ndm3p,ndm23p,ndm33p/
c
      data lintal,ldtnrm,lrerdr/.true.,.false.,.false./
c
c
      COMPLEX fstdet,fcn,EVSTORE
c
      external fcn
c
C     call link('unit3=(output,create,text),print3,unit59=ttv//')
c
C     CALL TEKALL(4014,120,0,0,0)
C     CALL BGNPL(0)
c
      print 100
      WRITE(3,100)
  100 format(1x,'first call to fcn initiated')
      fstdet=fcn(ev)
      EV=EVSTORE
      print 101
      WRITE(3,101)
  101 format(1x,'first call to fcn complete')
c
      if(neval.ne.0) goto 200
      print 102
      WRITE(3,102)
  102 format(1x,'call to cevalf initiated')
      ELF= cevalf(ev,fcn)
      print 103
      WRITE(3,103)
  103 format(1x,'call to cevalf complete')
```

```
c
c
      call nrmfcn
c
      call fltfcn
c
      goto 300
  200 continue
C     call endfl(0)
C     call donefl
      rewind 90
      write(90) neval
      do 10 nn=1,neval
      det=fcn(ev)
      write(90) ev,det
      CALL NRMFCN
      call fltfcn
      ev=ev+deval
10    continue
      endfile 90
  300 continue
      stop 999
      end
      subroutine sprnt (arin,irout,imax)
C     IMPLICIT REAL*8(A-H,O-Z)


c  input
c  arin:  beginning location of floating point numbers to process
c  irout:  beginning location to store compact bcd representation
c          of floating point numbers
c  imax:  total no. of successive floating point numbers to process

c  purpose
c  to convert n=imax successive floating point numbers into a compact
c  bcd representation in the e-format and store them in n successive
c  temporary locations beginning at irout which are later to be
c  output in r6 format.
c  for example the floating point number -1.7465e+25 is printed
c  as -175+q where the decimal point is assumed between the minus
c  sign and the digit 1.  the 2 digit exponent is converted to a single
c  character by a table lookup on (0,1,...,9,a,b,...z)=(0,1,2,...35)

c  if an i. or r. or *. is returned by subroutine etype, it is printed

c  if the exponent is greater than +35 or less than -35 a * is
c       printed


c.....
      REAL*16 aout

      dimension arin(2), irout(2), iex(73)

      data iex/4H  -*,4H  -z,4H  -y,4H  -x,4H  -w,4H  -v
     .    ,4H  -u,4H  -t,4H  -s
     .    ,4H  -r,4H  -q,4H  -p,4H  -o,4H  -n,4H  -m,4H  -l
     .    ,4H  -k,4H  -j,4H  -i,4H  -h
     .    ,4H  -g,4H  -f,4H  -e,4H  -d,4H  -c,4H  -b,4H  -a
     .    ,4H  -9,4H  -8,
     .    4H  -7,4H  -6,4H  -5,4H  -4,4H  -3,4H  -2
     .    ,4H  -1,4H  +0,4H  +1,4H  +2
     .    ,4H  +3,4H  +4,4H  +5,4H  +6,4H  +7,4H  +8,4H  +9
```

```
     .    ,4H  +a,4H  +b,4H  +c,4H  +d,4H  +e
     .    ,4H  +f,4H  +g,4H  +h,4H  +i,4H  +j,4H  +k,4H  +l
     .    ,4H  +m,4H  +n,4H  +o,4H  +p
     .    ,4H  +q,4H  +r,4H  +s,4H  +t,4H  +u,4H  +v,4H  +w
     .    ,4H  +x,4H  +y,4H  +z,4H  +*/


C      do 100 i = 1,imax
C    convert  arin(i) to e9.2 format
C      call zcetoa(aout,0,arin(i),9,2)
C    find exponent
C      call zciatob(aout,6,8,ind,2)
C    check if index within bounds.
C      ind=max0(min0(ind,35),-37)
C    pick up exponent.
C      irout(i) = iex(ind+38)
C    pick up sign bit and first significant digit.
C      call zmovechr(irout(i),0,aout,0,2)
C    pick up second and third significant digits.
C      call zmovechr(irout(i),2,aout,3,2)
C100   continue
       return
       end
       subroutine prnt (array,idim,jdim,imin,imax,jmin,jmax)
C      IMPLICIT REAL*8(A-H,O-Z)
       PARAMETER (icr = 19)
c..programmer: j breazeal
c..date: 1/10/75

c..things to consider in using this subroutine
c     set value of icr
c     set value of nout
c     add declarative for lcm if array is in lcm
c     add declarative  value idim,jdim,imin,imax,jmin,jmax
c     this routine calls sprnt (j breazeal) and orderlib routines

c..this routine is fashioned after the 'nrl' prnt routine (see d anderso

c..this routine will print the 2-d array specified.  a partial
c..printout of the array is obtained by setting imin,imax, and
c..jmin,jmax.  the maximum points per line is 19 in the i direction.
c..there is no limit in the j direction.  if more than 19 points
c..span the printout in the i direction, the printout occurs in
c..partial blocks, where i=imin to imin+19,for jmin to jmax;
c..              and next: i=imin+19 to imin+39,for jmin to jmax; etc.

c..usuage:

c      call prnt (array(1,1),idim,jdim,imin,imax,jmin,jmax)


c..array: array to be printed
c..idim:  dimension in i
c..jdim:  dimension in j
c..imin,imax: low and high range of index i
c..jmin,jmax: low and high range of index j
c..icr:  max no. of points to print per line


       PARAMETER (nout = 3)
       dimension array (idim,jdim),sparay(icr)
```

```
      i2 = 0
      do 25 iblk = imin,imax,icr

      i2 = i2 + icr + imin - 1
      if (i2.gt.imax) i2 = imax
      inum = i2 - iblk + 1
      write (nout,51) ((i,i=iblk,i2))

      do 20 j = jmin,jmax

        call sprnt (array(iblk,j),sparay,inum)
        write (nout,50) (j,(sparay(i),i=1,inum))
 20   continue

 25   continue

 50   format (3h j=,i2, 19a6)
 51   format (' i =' ,19i6)
      return
      end
c     subroutine leqt1c (a,n,ia,b,m,ib,ijob,wa,det,ier)
C     IMPLICIT REAL*8(A-H,O-Z)
c
c-leqt1c--------s/d-----library 1-----------------------------------
c
c    function              - matrix decomposition, linear equation
c                            solution - space economizer solution -
c                            COMPLEX matrices
c    usage                 - call leqt1c (a,n,ia,b,m,ib,ijob,wa,det,ier)
c    PARAMETERs    a       - input COMPLEX matrix of dimension n by n
c                            containing the COMPLEX coefficients of the
c                            equation ax = b.
c                          on output, a contains the l-u decomposition of
c                            a rowwise permutation of the input matrix a.
c                  n       - order of matrix a. (input)
c                  ia      - row dimension of a as specified in the calling
c                            program. (input)
c                  b       - input COMPLEX matrix of dimension n by m
c                            containing the m COMPLEX valued right hand
c                            sides of the equation ax = b.
c                          on output, the solution matrix x replaces b.
c                            if ijob=1, b is not used.
c                  m       - number of right hand sides (columns in b).
c                            (input)
c                  ib      - row dimension of b as specified in the
c                            calling program. (input)
c                  ijob    - input option PARAMETER.  ijob=i implies when
c                            i=0, factor the matrix and solve the
c                              equation ax=b.
c                            i=1, factor the matrix a.
c                            i=2, solve the equation ax=b.  this
c                              option implies that leqt1c has already
c                              been called using ijob=0 or 1 so that
c                              the matrix has already been factored.  in
c                              this case output matrix a must have been
c                              saved for reuse in the call to leqt1c.
c                  wa      - work area of length n containing the pivot
c                            indices.
cm                 det     - determinant of matrix a.
c                  ier     - error PARAMETER.
c                            terminal error = 128+n
c                              n=1 indicates that matrix a is
c                                algorithmically singular.  (see the
```

```
c                                chapter 1 prelude).
c    precision           - single/double
c    reqd. imsl routines - uertst
c    language            - fortran
c-----------------------------------------------------------------------
c    latest revision     - january 8,1973
cm
cm   addition of determinant calculation  -  may 17,1979
cm
c
      subroutine leqt1c (a,n,ia,b,m,ib,ijob,wa,det,ier)
C     IMPLICIT REAL*8(A-H,O-Z)
c
      dimension          a(ia,1),b(ib,1),wa(1),t(2)
c*    double precision   p,q,zero,one,wa,t,rn,bis
c1    COMPLEX          a,b,sum,temp
      COMPLEX            a,b,sum,temp,det
      equivalence        (sum,t(1))
c1    data               zero/0.d0/,one/1.d0/
      data               zero/0.0/,one/1.0/
cm
c                                      initialization
      ier = 0
      if (ijob .eq. 2) go to 75
      rn = n
c                                      find equilibration factors
      do 10 i=1,n
         bis = zero
         do 5 j=1,n
            temp = a(i,j)
c1          p = cdabs(temp)
            p = ABS(temp)
            if (p .gt. bis) bis = p
cm
      det = (1.,0.)
      sgn = 1.
    5    continue
         if (bis .eq. zero) go to 105
         wa(i) = one/bis
   10 continue
c                                      l-u decomposition
      do 70 j = 1,n
         jm1 = j-1
         if (jm1 .lt. 1) go to 25
c                                      compute u(i,j), i=1,...,j-1
         do 20 i=1,jm1
            sum = a(i,j)
            im1 = i-1
            if (im1 .lt. 1) go to 20
            do 15 k=1,im1
               sum = sum-a(i,k)*a(k,j)
   15       continue
            a(i,j) = sum
   20    continue
   25    p = zero
c                                      compute u(j,j) and l(i,j), i=j+1,...,
         do 45 i=j,n
            sum = a(i,j)
            if (jm1 .lt. 1) go to 40
            do 35 k=1,jm1
               sum = sum-a(i,k)*a(k,j)
   35       continue
            a(i,j) = sum
```

```
c1 40        q = wa(i)*cdabs(sum)
   40        q = wa(i)*ABS(sum)
             if (p .ge. q) go to 45
             p = q
             imax = i
   45     continue
c                                    test for algorithmic singularity
          if (rn+p .eq. rn) go to 105
          if (j .eq. imax) go to 60
c                                    interchange rows j and imax
          ssn = -1.*ssn
cm
          do 50 k=1,n
             temp = a(imax,k)
             a(imax,k) = a(j,k)
             a(j,k) = temp
   50     continue
          wa(imax) = wa(j)
   60     wa(j) = imax
          jp1 = j+1
          if (jp1 .gt. n) go to 70
c                                    divide by pivot element u(j,j)
          temp = a(j,j)
          do 65 i = jp1,n
             a(i,j) = a(i,j)/temp
   65     continue
   70 continue
cm                              calculate determinant
       do 72 i=1,n
          det = det*a(i,i)
   72 continue
cm
          det = ssn*det
cm
   75 if (ijob .eq. 1) go to 9005
      do 103 k = 1,m
c                                    solve ux = y for x
          iw = 0
          do 90 i = 1,n
             imax = wa(i)
             sum = b(imax,k)
             b(imax,k) = b(i,k)
             if (iw .eq. 0) go to 85
             im1 = i-1
             do 80 j = iw,im1
                sum = sum-a(i,j)*b(j,k)
   80        continue
             go to 88
   85        if (t(1) .ne. zero .or. t(2) .ne. zero) iw = i
   88        b(i,k) = sum
   90     continue
c                                    solve ly = b for y
          n1 = n+1
          do 100 iw = 1,n
             i = n1-iw
             jp1 = i+1
             sum = b(i,k)
             if (jp1 .gt. n) go to 98
             do 95 j = jp1,n
                sum = sum-a(i,j)*b(j,k)
   95        continue
   98        b(i,k) = sum/a(i,i)
  100     continue
```

```
  103 continue
      go to 9005
c                                     algorithmic singularity
  105 ier = 129
 9000 continue
c                                     print error
      call uertst(ier,6hleqt1c)
 9005 return
      end
c     subroutine uertst (ier,name)
C     IMPLICIT REAL*8(A-H,O-Z)
c
c-uertst----------------library 1-----------------------------------
c
c    function              - error message generation
c    usage                 - call uertst(ier,name)
c    PARAMETERs    ier     - error PARAMETER. type + n  where
c                              type= 128 implies terminal error
c                                    64 implies warning with fix
c                                    32 implies warning
c                              n   = error code relevant to calling routine
c                  name    - input vector containing the name of the
c                              calling routine as a six character literal
c                              string.
c    language              - fortran
c-----------------------------------------------------------------------
c    latest revision       - January 18, 1974
c
      subroutine uertst(ier,name)
C     IMPLICIT REAL*8(A-H,O-Z)
c
      dimension            ityp(5,4),ibit(4)
      integer              name(3)
      integer              warn,warf,term,printr
      equivalence          (ibit(1),warn),(ibit(2),warf),(ibit(3),terr)
      data       ityp      /'warn','ing ',',    ',',    ',',    ',
     *                      'warn','ing(','with',' fix',')    ',
     *                      'term','inal',',    ',',    ',',    ',
     *                      'non-','defi','ned ',',    ',',    '/,
     *           ibit       / 32,64,128,0/
      data       printr     / 3/
      ier2=ier
      if (ier2 .ge. warn) go to 5
c                                     non-defined
      ier1=4
      go to 20
    5 if (ier2 .lt. term) go to 10
c                                     terminal
      ier1=3
      go to 20
   10 if (ier2 .lt. warf) go to 15
c                                     warning(with fix)
      ier1=2
      go to 20
c                                     warning
   15 ier1=1
c                                     extract 'n'
   20 ier2=ier2-ibit(ier1)
c                                     print error message
      write (printr,25) (ityp(i,ier1),i=1,5),name,ier2,ier
   25 format(' *** i m s l(uertst) ***  ',5a4,4x,a4,a2,4x,i2,
     *    ' (ier = ',i3,')')
      return
```

```
      end
      subroutine bsplcf
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c---routine computes the coefficients of the b-splines which form
c   a basis over the set of knots, r, with repeated knots
c   at the end points.  s(j,l,i) j-power, l-segment, i-spline.
c---
c
c
      PARAMETER(NP  = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/ni,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
c---i=1
      c4=4.00/((r(2)-r(1))**4)
      idx=16
      sa(idx)   =              -c4
      sa(idx-1) =           3.0*c4*r(2)
      sa(idx-2) =          -3.0*c4*r(2)*r(2)
```

```
      sa(idx-3) =                  c4*r(2)*r(2)*r(2)
c---i=2
      c4=4.0/((r(3)-r(2))*(r(3)-r(1))**3)
      c3=4.0/((r(2)-r(3))*(r(2)-r(1))**3)
      idx=32
      sa(idx)   =                 -c4
      sa(idx-1) =              3.0*c4*r(3)
      sa(idx-2) =             -3.0*c4*r(3)*r(3)
      sa(idx-3) =                   c4*r(3)*r(3)*r(3)
      sa(idx-4) =sa(idx)          -c3
      sa(idx-5) =sa(idx-1)  +3.0*c3*r(2)
      sa(idx-6) =sa(idx-2)  -3.0*c3*r(2)*r(2)
      sa(idx-7) =sa(idx-3)       +c3*r(2)*r(2)*r(2)
c---i=3
      c4=4.0/((r(4)-r(2))*(r(4)-r(3))*(r(4)-r(1))**2)
      c3=4.0/((r(3)-r(2))*(r(3)-r(4))*(r(3)-r(1))**2)
      c2=4.0/((r(2)-r(3))*(r(2)-r(4))*(r(2)-r(1))**2)
      idx=48
      sa(idx)   =                 -c4
      sa(idx-1) =             +3.0*c4*r(4)
      sa(idx-2) =             -3.0*c4*r(4)*r(4)
      sa(idx-3) =                   c4*r(4)*r(4)*r(4)
      sa(idx-4) =sa(idx)          -c3
      sa(idx-5) =sa(idx-1)  +3.0*c3*r(3)
      sa(idx-6) =sa(idx-2)  -3.0*c3*r(3)*r(3)
      sa(idx-7) =sa(idx-3)       +c3*r(3)*r(3)*r(3)
      sa(idx-8) =sa(idx-4)       -c2
      sa(idx-9) =sa(idx-5)  +3.0*c2*r(2)
      sa(idx-10)=sa(idx-6)  -3.0*c2*r(2)*r(2)
      sa(idx-11)=sa(idx-7)       +c2*r(2)*r(2)*r(2)
c---i=4,n-1
      do 10 i=4,nm1
      m1=i-3
      m2=i-2
      m3=i-1
      m4=i
      m5=i+1
      c4=4.0/((r(m5)-r(m1))*(r(m5)-r(m2))*(r(m5)-r(m3))*(r(m5)-r(m4)))
      c3=4.0/((r(m4)-r(m1))*(r(m4)-r(m2))*(r(m4)-r(m3))*(r(m4)-r(m5)))
      c2=4.0/((r(m3)-r(m1))*(r(m3)-r(m2))*(r(m3)-r(m4))*(r(m3)-r(m5)))
      c1=4.0/((r(m2)-r(m1))*(r(m2)-r(m3))*(r(m2)-r(m4))*(r(m2)-r(m5)))
      idx=16*i
      sa(idx)   =                 -c4
      sa(idx-1) =             +3.0*c4*r(m5)
      sa(idx-2) =             -3.0*c4*r(m5)*r(m5)
```

```
      sa(idx-3) =                  c4*r(m5)*r(m5)*r(m5)
```

```
      sa(idx-4) =sa(idx)        -c3
      sa(idx-5) =sa(idx-1) +3.0*c3*r(m4)
      sa(idx-6) =sa(idx-2) -3.0*c3*r(m4)*r(m4)
      sa(idx-7) =sa(idx-3)      +c3*r(m4)*r(m4)*r(m4)
      sa(idx-8) =sa(idx-4)      -c2
      sa(idx-9) =sa(idx-5) +3.0*c2*r(m3)
      sa(idx-10)=sa(idx-6) -3.0*c2*r(m3)*r(m3)
      sa(idx-11)=sa(idx-7)      +c2*r(m3)*r(m3)*r(m3)
      sa(idx-12)=sa(idx-8)      -c1
      sa(idx-13)=sa(idx-9) +3.0*c1*r(m2)
      sa(idx-14)=sa(idx-10)-3.0*c1*r(m2)*r(m2)
      sa(idx-15)=sa(idx-11)     +c1*r(m2)*r(m2)*r(m2)
   10 continue
c---i=n
      m1=n-3
      m2=n-2
      m3=n-1
      m4=n
      c4=12.0/((r(m4)-r(m1))*(r(m4)-r(m2))*(r(m4)-r(m3)))
      c3=-4.0/((r(m4)-r(m2))*(r(m4)-r(m3))*(r(m4)-r(m1))**2)
     2   -4.0/((r(m4)-r(m1))*(r(m4)-r(m3))*(r(m4)-r(m2))**2)
     3   -4.0/((r(m4)-r(m1))*(r(m4)-r(m2))*(r(m4)-r(m3))**2)
      c2= 4.0/((r(m3)-r(m1))*(r(m3)-r(m2))*(r(m3)-r(m4))**2)
      c1= 4.0/((r(m2)-r(m1))*(r(m2)-r(m3))*(r(m2)-r(m4))**2)
      idx=16*n
      sa(idx-4) =               -c3
      sa(idx-5) =c4          +3.0*c3*r(m4)
      sa(idx-6) = -2.0*c4    -3.0*c3*r(m4)*r(m4)
      sa(idx-7) =c4*r(m4)*r(m4)+c3*r(m4)*r(m4)*r(m4)
      sa(idx-8) =sa(idx-4)      -c2
      sa(idx-9) =sa(idx-5) +3.0*c2*r(m3)
      sa(idx-10)=sa(idx-6) -3.0*c2*r(m3)*r(m3)
      sa(idx-11)=sa(idx-7)      +c2*r(m3)*r(m3)*r(m3)
      sa(idx-12)=sa(idx-8)      -c1
      sa(idx-13)=sa(idx-9) +3.0*c1*r(m2)
      sa(idx-14)=sa(idx-10)-3.0*c1*r(m2)*r(m2)
      sa(idx-15)=sa(idx-11)     +c1*r(m2)*r(m2)*r(m2)
c---i=n+1
      c4= 12.0/((r(m4)-r(m2))     *(r(m4)-r(m3)))
      c3=-12.0/((r(m4)-r(m3))     *(r(m4)-r(m2))**2)
     2   -12.0/((r(m4)-r(m2))     *(r(m4)-r(m3))**2)
      c2=  4.0/((r(m4)-r(m3))     *(r(m4)-r(m2))**3)
     2    +4.0/((r(m4)-r(m2))**2 *(r(m4)-r(m3))**2)
     3    +4.0/((r(m4)-r(m2))     *(r(m4)-r(m3))**3)
      c1=  4.0/((r(m3)-r(m2))     *(r(m3)-r(m4))**3)
      idx=16*(n+1)
      sa(idx-8) =                          -c2
      sa(idx-9) =           c3          +3.0*c2*r(m4)
      sa(idx-10)=-c4  -2.0*c3*r(m4)   -3.0*c2*r(m4)*r(m4)
      sa(idx-11)= c4*r(m4)+c3*r(m4)*r(m4)+c2*r(m4)*r(m4)*r(m4)
      sa(idx-12)=sa(idx-8)      -c1
      sa(idx-13)=sa(idx-9) +3.0*c1*r(m3)
      sa(idx-14)=sa(idx-10)-3.0*c1*r(m3)*r(m3)
      sa(idx-15)=sa(idx-11)     +c1*r(m3)*r(m3)*r(m3)
c---i=n+2
      c4= +4.0/(r(m4)-r(m3))
      c3=-12.0/(r(m4)-r(m3))**2
      c2=+12.0/(r(m4)-r(m3))**3
      c1= -4.0/(r(m4)-r(m3))**4
      idx=16*(n+2)
      sa(idx-12)=                          -c1
      sa(idx-13)=             c2          +3.0*c1*r(m4)
      sa(idx-14)=  -c3  -2.0*c2*r(m4)   -3.0*c1*r(m4)*r(m4)
```

```
      sa(idx-15)=c4+c3*r(m4)+c2*r(m4)*r(m4)+c1*r(m4)*r(m4)*r(m4)
      return
      end
      subroutine depse(fn,c)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley     univ. of texas at austin  jan 1976
      dimension fn(5),c(1)
c
c
      PARAMETER(NP   = 21,
     2          nesp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          nesqp=nesp*nesp,nep=4*nesp+1,
     3          maxp=nesp*nsdp,nsap=16*np2p,na1p=7*nesqp,
     4          ndm2p=nesp*np2p,ndm3p=nesqp*np2p,
     5          ndm23p=3*nesp,ndm33p=3*nesqp)
c
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      nm3=n-3
      if(abs(fn(1)).gt.abs(10.*fn(2))) go to 20
c---find derivative of fcn at end pts.
       d11=(fn(2)-fn(1))/(r(2)-r(1))
       d12=(fn(3)-fn(2))/(r(3)-r(2))
       d13=(fn(4)-fn(3))/(r(4)-r(3))
       d21=(d12-d11)/(r(3)-r(1))
```

```
        d22=(d13-d12)/(r(4)-r(2))
        d31=(d22-d21)/(r(4)-r(1))
       fp0=d11-d21*r(2)+d31*r(2)*r(3)
c---
        d11=(fn(nm2)-fn(nm3))/(r(nm2)-r(nm3))
        d12=(fn(nm1)-fn(nm2))/(r(nm1)-r(nm2))
        d13=(fn(n)-fn(nm1))/(r(n)-r(nm1))
        d21=(d12-d11)/(r(nm1)-r(nm3))
        d22=(d13-d12)/(r(n)-r(nm2))
        d31=(d22-d21)/(r(n)-r(nm3))
        fp1=+r(n)*(d13+(r(n)-r(nm1))*(d22+d31*(r(n)-r(nm2))))
c---compute e and f.
       c(1)=fn(1)/p01(1)
    22 continue
       f(2)=(fp0-p11(1)*c(1))/p12(1)
       e(2)=0.0
       e(3)=-p03(2)/p02(2)
       f(3)=(fn(2)-p01(2)*f(2))/p02(2)
       do 10 i=4,n
        e(i)=1.0/(p01(i-1)*e(i-1)+p02(i-1))
        f(i)=(fn(i-1)-p01(i-1)*f(i-1))*e(i)
        e(i)=-p03(i-1)*e(i)
    10 continue
c---compute c.
       c(np2)=fn(n)/p03(n)
       c(np1)=(fp1-p13(n)*c(np2))/p12(n)
        do 12 i=2,n
        j=np2-i
        c(j)=e(j)*c(j+1)+f(j)
    12 continue
       return
    20 continue
        d11=(fn(3)-fn(2))/(r(3)-r(2))
        d12=(fn(4)-fn(3))/(r(4)-r(3))
        d13=(fn(5)-fn(4))/(r(5)-r(4))
        d21=(d12-d11)/(r(4)-r(2))
        d22=(d13-d12)/(r(5)-r(3))
        d31=(d22-d21)/(r(5)-r(2))
       fn0=fn(2)+(r(1)-r(2))*(d11+(r(1)-r(3))*(d21
      2                          +(r(1)-r(4))* d31))
       fp0=d11+d21*((r(1)-r(3))+(r(1)-r(2)))+d31*((r(1)-r(3))*
      2    (r(1)-r(4))+(r(1)-r(2))*(r(1)-r(4))+
      3 (r(1)-r(2))*(r(1)-r(3)))
       c(1)=fn0/p01(1)
       go to 22
       end
       subroutine deset(jh)
C      IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c
c
       PARAMETER(NP   = 21,
      2          neqp=  1,
      3          nsip=  1,
      4          ndip=  1,
      5          ntip=  4,
      6          ngdp=101,
      7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
      7          neqsqp=neqp*neqp,nep=4*neqp+1,
      3          maxp=neqp*ngdp,nsap=16*np2p,na1p=7*neqsqp,
      4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
      5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
```

```
c
c
      common/ncl/nm2,nm1,n,np1,np2
      common/rcl/r(np),rn(np)
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      data pi1(1),pi2(1),pi3(1),pi4(1)/4*0.0/
c---subroutine sets up values of splines at the knots.
      do 10 i=1,n
      call spvl(i,r(i),p01(i),1)
      call spvl(i+1,r(i),p02(i),1)
      call spvl(i+2,r(i),p03(i),1)
      call spvlp(i,r(i),p11(i),1)
      call spvlp(i+1,r(i),p12(i),1)
      call spvlp(i+2,r(i),p13(i),1)
      call spvlpp(i,r(i),p21(i),1)
      call spvlpp(i+1,r(i),p22(i),1)
      call spvlpp(i+2,r(i),p23(i),1)
   10 continue
      p02(1)=0.0
      p03(1)=0.0
      p01(n)=0.0
      p02(n)=0.0
      p13(1)=0.0
      p11(n)=0.0
      do 12 i=2,n
      pi1(i)=saus6(i-1,i,jh)
      pi2(i)=saus6(i,i,jh)
      pi3(i)=saus6(i+1,i,jh)
      pi4(i)=saus6(i+2,i,jh)
   12 continue
      bc0p  = p11(1)
      bc0p1 = p12(1)
      bc1p1 = p12(n)
```

```
      bc1p   = p13(n)
      bc1    = p03(n)
      bc0    = p01(1)
      return
      end
       function saus10(p1,p2,p3,n,a,b)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
      dimension p1(5),p2(5),p3(5)
      dataw1,w2,w3/.467913934572691,.360761573048139,.171324492379170/
      datax1,x2,x3/.238619186083197,.661209386466265,.932469514203152/
      fcn(x)=(p1(1)+x*(p1(2)+x*(p1(3)+x*(p1(4)+x*p1(5)))))*
     2       (p2(1)+x*(p2(2)+x*(p2(3)+x*(p2(4)+x*p2(5)))))*
     3       (p3(1)+x*(p3(2)+x*(p3(3)+x*(p3(4)+x*p3(5)))))*
     4       (x**n)
      rd=0.5*(b-a)
      rp=0.5*(b+a)
      saus10 =rd*(w3*(fcn(rp-rd*x3)+fcn(rp+rd*x3))
     2           +w2*(fcn(rp-rd*x2)+fcn(rp+rd*x2))
     3           +w1*(fcn(rp-rd*x1)+fcn(rp+rd*x1)))
      return
      end
       function saus6(k,i,jh)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
c
       common/nc1/nm2,nm1,n,np1,np2
       common/rc1/r(np),rn(np)
       common/sac1/sa(nsap)
       common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
       common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
       common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
       common/phii1/pi1(np)/phii2/pi2(np)
       common/phii3/pi3(np)/phii4/pi4(np)
       common/phi4/e(np2p)/phi5/f(np2p)
       common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
       common/errc1/err(np2p)
       common/serrc1/amxer1,amxer2,ermax
       common/dum1/d1(np2p)/dum2/d2(np2p)
c
c      commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
       common/ssic1/ssi(np2p,nsip)
```

```
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      data w1,w2,w3/.467913934572691,.360761573048139,.171324492379170/
      data x1,x2,x3/.238619186083197,.661209386466265,.932469514203152/
      dimension y(6),x(6)
      rd=0.5*(r(i)-r(i-1))
      rp=0.5*(r(i)+r(i-1))
      x(1)=rp-rd*x3
      x(2)=rp-rd*x2
      x(3)=rp-rd*x1
      x(4)=rp+rd*x1
      x(5)=rp+rd*x2
      x(6)=rp+rd*x3
      call spvl(k,x(1),y(1),6)
      saus6=rd*(w3*(x(1)**jh*y(1)+x(6)**jh*y(6))
     2         +w2*(x(2)**jh*y(2)+x(5)**jh*y(5))
     3         +w1*(x(3)**jh*y(3)+x(4)**jh*y(4)))
      return
      end
      subroutine grid(er,isw)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c
c
      PARAMETER(NP  = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
c
      common/ncl/nm2,nm1,n,np1,np2
      common/rcl/r(np),rn(np)
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline intesrals
c
```

```
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      dimension er(1)
c---grid sets up grid spacing.
c     isw=1,uniform grid.
c     isw=2,spacing based on err fcn.
c     isw=3, >    >>>>>>>
c     isw=4,uniform except end pts.
      go to (801,802,803,804),isw
c---sets up uniform mesh.
  801 do 10 i=1,n
   10 er(i)=(i-1.0)/(n-1.0)
      return
c---
c---this section choses a new set of knots based on the error
c   function er.
c---
c---note: er on exit contains new x.
  802 continue
      emax=0.0
      do 19 i=2,nm1
   19 emax=MAX(emax,er(i))
      emin=.001*emax
      do 20 i=2,nm1
      er(i)=MAX(emin,er(i))
      er(i)=er(i)**0.25
   20 continue
      er(1)=er(2)
      er(n)=er(nm1)
      sum=0.0
      do 21 i=1,nm1
   21 sum=sum+0.5*(er(i)+er(i+1))*(r(i+1)-r(i))
      sum=sum/nm1
c---compute partition.
      rn(1)=0.0
      k=1
      tot=0.0
      iflg=1
      do 22 i=2,nm1
      goal=(i-1)*sum
   25 del=(r(k+1)-r(k))
      add=del*(er(k+1)+er(k))*0.5
      if(add+tot.lt.goal) go to 23
      rn(i)=r(k)+(goal-tot)*del/add
      go to 22
   23 tot=tot+add
      k=k+1
      go to 25
```

```
   22 continue
      rn(1)=0.0
      rn(n)=1.0
      do 33 i=1,n
      er(i)=rn(i)
   33 continue
      return
  803 continue
c---map er to scaling form.
      er(1)=er(2)
      er(n)=er(nm1)
      do 40 i=1,n
      if(er(i).gt.amxer2) go to 41
      if(er(i).gt.amxer1)go to 42
      if(er(i).gt.0.2*amxer1) go to 43
      er(i)=1.25
      go to 40
   41 er(i)=0.60
      go to 40
   42 er(i)=0.75
      go to 40
   43 er(i)=1.00
   40 continue
c---compute new r.
      rin=0.0
      do 50 i=1,nm1
      rip1n=rin+(r(i+1)-r(i))*0.5*(er(i)+er(i+1))
      er(i)=rin
      rin=rip1n
   50 continue
      er(n)=rip1n
c---rescale.
      do 51 i=1,n
   51 er(i)=er(i)/er(n)
c---check er for minimun spacing.
      do 54 i=2,n
      if(er(i)-er(i-1).gt..01) go to 54
      er(i)=er(i-1)+.01
   54 continue
      do 55 i=2,n
   55 er(i)=er(i)/er(n)
      return
  804 continue
      nm4=n-4
      dlr=1./(nm4-1)
      er(1)=0.0
      er(2)=.5*dlr
      er(3)=dlr
      er(4)=1.5*dlr
      do 87 i=5,nm4
      er(i)=(i-3)*dlr
   87 continue
      er(n-3)=(nm4-2.5)*dlr
      er(n-2)=(nm4-2)*dlr
      er(n-1)=(nm4-1.5)*dlr
      er(n)=1.
      return
      end
      subroutine pop(a,p,l,n,ns)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
      dimension p(5),a(4)
      imax=4+n-1
```

```
      do 10 i=1,5
   10 p(i)=0.0
      jmin=max0(1+l-n,1)
      ns=max0(1-jmin+l,0)
      if(jmin.gt.4) return
      do 11 j=jmin,4
      fac=1.0
      if(l.eq.0) go to 12
      do 13 il=1,l
   13 fac=(j+n-1-l+il)*fac
   12 p(j+ns-1)=fac*a(j)
   11 continue
      return
      end
      subroutine reordr(u,nk,rnew)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c---reordr interpolates u from current grid,r, to new grid.
c
      PARAMETER(NP  = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
      dimension rnew(1),u(np,nk)
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
```

```
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
      dimension ui(1),sp(1)
c
c
c---
      do 12 k=1,nk
      call depse(u(1,k),ui)
      do 13 i=1,n
   13 u(i,k)=0.0
      do 12 l=1,np2
      call spvl(l,rnew,sp,n)
      do 12 i=1,n
      u(i,k)=u(i,k)+ui(l)*sp(i)
   12 continue
      return
      end
      subroutine repse(c,v)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
      dimension c(1),v(1)
c
      PARAMETER(NP   = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsp,
     4          ndm2p=neqp*np2p,ndm3p=neqsp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsp)
c
c
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
```

```
      common/nssiP/nsi,nsJ(nsiP),nsv(2,nsiP)
      common/ndsiP/ndi,ndJ(ndiP),ndv(4,ndiP)
      common/ntsiP/nti,ntJ(ntiP),ntv(6,ntiP)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,nP2P)
c
c
c---computes the function values at the test points from the spline coef
      do 10 i=1,n
      v(i)=c(i)*P01(i)+c(i+1)*P02(i)+c(i+2)*P03(i)
   10 continue
      return
      entry repsP(c,v)
      do 11 i=1,n
      v(i)=c(i)*P11(i)+c(i+1)*P12(i)+c(i+2)*P13(i)
   11 continue
      return
      entry repsPP(c,v)
      do 12 i=1,n
      v(i)=c(i)*P21(i)+c(i+1)*P22(i)+c(i+2)*P23(i)
   12 continue
      return
      entry repsi(c,v)
      v(1)=0.0
      do 13 i=2,n
      v(i)=v(i-1)+c(i-1)*Pi1(i)+c(i)*Pi2(i)+c(i+1)*Pi3(i)+
     2           c(i+2)*Pi4(i)
   13 continue
      return
      end
      subroutine rmove(rnew)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by J. C. wiley    univ. of texas at austin  Jan 1976
c
c
      PARAMETER(NP   = 21,
     2          neqP=  1,
     3          nsiP=  1,
     4          ndiP=  1,
     5          ntiP=  4,
     6          nsdP=101,
     7          nm2P=nP-2,nm1P=nP-1,nP1P=nP+1,nP2P=nP+2,
     7          neqsqP=neqP*neqP,nep=4*neqP+1,
     3          maxP=neqP*nsdP,nsaP=16*nP2P,na1P=7*neqsqP,
     4          ndm2P=neqP*nP2P,ndm3P=neqsqP*nP2P,
```

```
     5           ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline intesrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of sinsle intesrals
c#### j=number of double intesrals
c#### k=number of triple intesrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      dimension rnew(1)
      do 10 i=1,n
      r(i)=rnew(i)
      rnew(i)=0.0
   10 continue
      return
      end
      subroutine sfeval
C     IMPLICIT REAL*8(A-H,O-Z)
c
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c
c
      PARAMETER(NP  = 21,
     2          neqp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          neqsqp=neqp*neqp,nep=4*neqp+1,
     3          maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
     4          ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
     5          ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
```

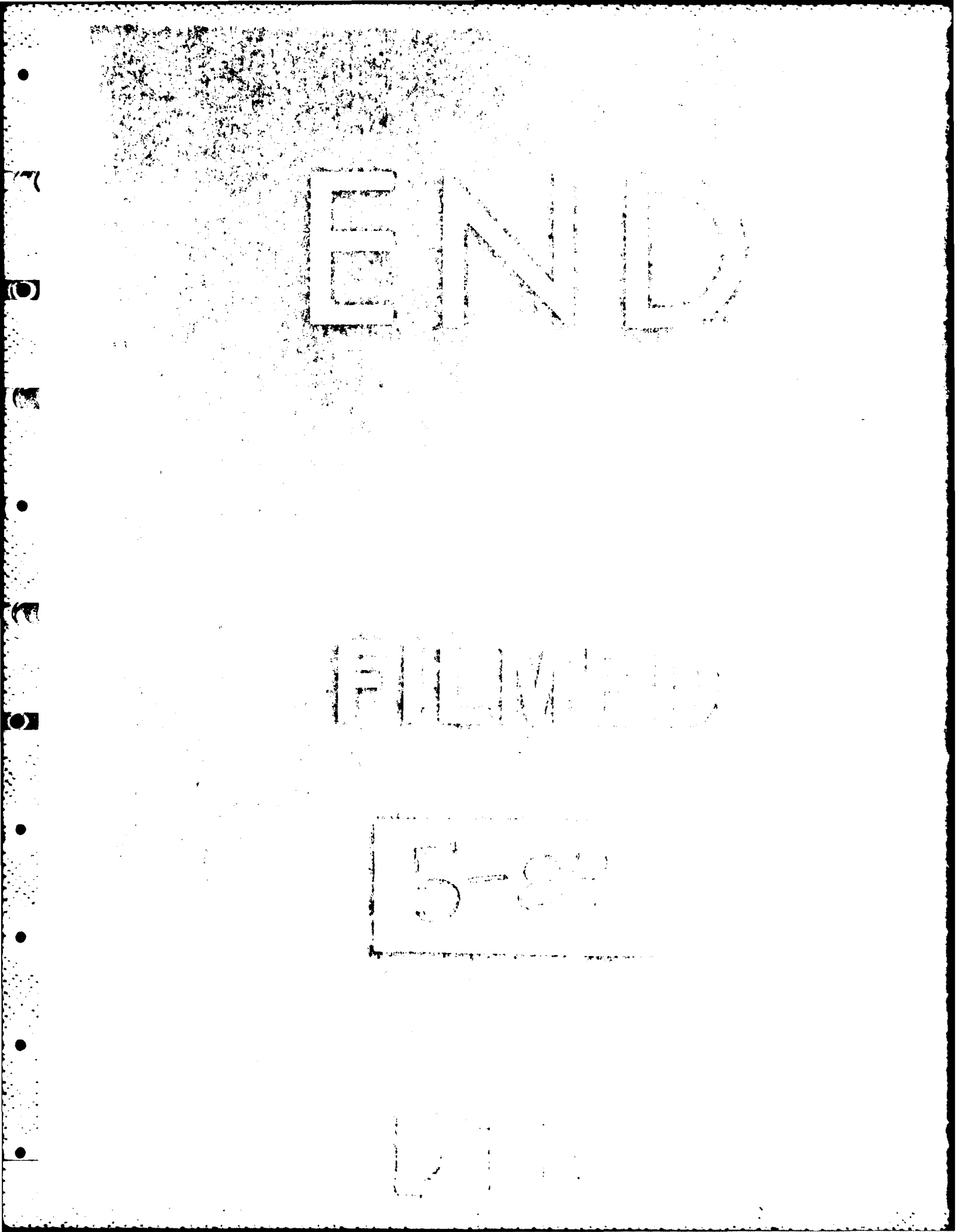
```
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
      common/sac1/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errc1/err(np2p)
      common/serrc1/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssic1/ssi(np2p,nsip)
      common/dsic1/dsi(np2p,7,ndip)
      common/tsic1/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      dimension p1(5),p2(5),p3(5)
      np3=np2+1
c---single spline integrals=
      do 8 i=1,5
      p2(i)=0.0
    8 p3(i)=0.0
      p2(1)=1.
      p3(1)=1.
      if(nsi.eq.0) go to 20
      do 10 nsiv=1,nsi
      do 10 i=1,np2
      lmin=max0(5-i,1)
      lmax=min0(np3-i,4)
      sum=0.0
      do 11 l=lmin,lmax
      idx=4*l+16*i-19
      call pop(sa(idx),p1,nsv(1,nsiv),nsv(2,nsiv),ns1)
   11 sum=sum+saus10(p1,p2,p3,nsj(nsiv)-ns1,r(i+l-4),r(i+l-3))
   10 ssi(i,nsiv)=sum
c---double spline integrals.
   20 if(ndi.eq.0) go to 30
      do 19 i=1,5
   19 p3(i)=0.0
      p3(1)=1.0
      do 29 ndiv=1,ndi
      do 29 i=1,np2
      do 29 j=1,7
      ij=i+j-4
      dsi(i,j,ndiv)=0.0
      if(ij.lt.1.or.ij.gt.np2) go to 29
```

```
      lmin=max0(max0(1,5-i),max0(1,5-ij)+j-4)
      lmax=min0(min0(4,np3-i),min0(4,np3-ij)+j-4)
      sum=0.0
      do 21 l=lmin,lmax
      idx=4*l+16*i-19
      idxk=4*(l+4-j)+16*(i+j-4)-19
      call bsp(sa(idx),p1,ndv(1,ndiv),ndv(2,ndiv),ns1)
      call bsp(sa(idxk),p2,ndv(3,ndiv),ndv(4,ndiv),ns2)
   21 sum=sum+saus10(p1,p2,p3,ndj(ndiv)-ns1-ns2,r(i+l-4),r(i+l-3))
   29 dsi(i,j,ndiv)=sum
c---triple spline integrals.
   30 if(nti.eq.0) return
      do 31 ntiv=1,nti
      do 31 i=1,np2
      do 31 j=1,7
      do 31 k=1,7
      ij=i+j-4
      ik=i+k-4
      idx=j+(k-1)*7
      tsi(i,idx,ntiv)=0.0
      if(ij.lt.1.or.ij.gt.np2) go to 31
      if(ik.lt.1.or.ik.gt.np2) go to 31
      lmin=max0(max0(1,5-i),max0(1,5-ij)+j-4,max0(1,5-ik)+k-4)
      lmax=min0(min0(4,np3-i),min0(4,np3-ij)+j-4,min0(4,np3-ik)+k-4)
      sum=0.0
      if(lmin.gt.lmax) go to 31
      do 32 l=lmin,lmax
      idx1=4*l+16*i-19
      idx2=4*(l+4-j)+16*ij-19
      idx3=4*(l+4-k)+16*ik-19
      call bsp(sa(idx1),p1,ntv(1,ntiv),ntv(2,ntiv),ns1)
      call bsp(sa(idx2),p2,ntv(3,ntiv),ntv(4,ntiv),ns2)
      call bsp(sa(idx3),p3,ntv(5,ntiv),ntv(6,ntiv),ns3)
   32 sum=sum+saus10(p1,p2,p3,ntj(ntiv)-ns1-ns2-ns3,r(i+l-4),r(i+l-3))
      tsi(i,idx,ntiv)=sum
   31 continue
      return
      end
      subroutine splerr(c,er)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
      dimension er(1),c(1)
c---relative error estimate of spline fit.
c---note: the routine only returns a value in er(i) if
c---       the error is greater than the initial value of er(i).
c---note: if er is used with routine grid, grid zeroes er.
c
c
      PARAMETER(NP   = 21,
     2          nesp=  1,
     3          nsip=  1,
     4          ndip=  1,
     5          ntip=  4,
     6          nsdp=101,
     7          nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
     7          nesqp=nesp*nesp,nep=4*nesp+1,
     3          maxp=nesp*nsdp,nsap=16*np2p,na1p=7*nesqp,
     4          ndm2p=nesp*np2p,ndm3p=nesqp*np2p,
     5          ndm23p=3*nesp,ndm33p=3*nesqp)
c
c
      common/nc1/nm2,nm1,n,np1,np2
      common/rc1/r(np),rn(np)
```

```
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      fcn(i,l,x)=sa3(1,l,i)+x*(sa3(2,l,i)+x*(sa3(3,l,i)+x*sa3(4,l,i)))
      fval=0.0
      do 11 i=1,n
      fval=fval+abs(c(i)*fcn(i,4,r(i))+c(i+1)*fcn(i+1,3,r(i))+
     2 c(i+2)*fcn(i+2,2,r(i)))
   11 continue
      fval=fval/n
      if(fval.eq.0.0) fval=1.
      do 10 i=2,nm1
      error=0.02625*(c(i-1)*(            -sa3(4,4,i-1))
     2              +c(i)  *(sa3(4,4,i)  -sa3(4,3,i)  )
     3              +c(i+1)*(sa3(4,3,i+1)-sa3(4,2,i+1))
     4              +c(i+2)*(sa3(4,2,i+2)-sa3(4,1,i+2))
     5              +c(i+3)*(sa3(4,1,i+3)))
     6         *(MAX(r(i+1)-r(i),r(i)-r(i-1))**3)
      error=abs(error/fval)
      if(error.gt.er(i))er(i)=error
   10 continue
      return
      end
      subroutine spvl(i,x,v,m)
C     IMPLICIT REAL*8(A-H,O-Z)
c***** written by j. c. wiley    univ. of texas at austin  jan 1976
c---computes m values of the i-th b-spline at the m x-values and
c   returns them in v. note that the x-values are assumed to be
c---ordered.
c---
      dimension x(m),v(m)
c
c
      PARAMETER(NP   = 21,
     2          nsap=  1,
```

```
      3           nsip=  1,
      4           ndip=  1,
      5           ntip=  4,
      6           nsdp=101,
      7           nm2p=np-2,nm1p=np-1,np1p=np+1,np2p=np+2,
      7           neqsqp=neqp*neqp,nep=4*neqsqp+1,
      3           maxp=neqp*nsdp,nsap=16*np2p,na1p=7*neqsqp,
      4           ndm2p=neqp*np2p,ndm3p=neqsqp*np2p,
      5           ndm23p=3*neqp,ndm33p=3*neqsqp)
c
c
c
      common/ncl/nm2,nm1,n,np1,np2
      common/rcl/r(np),rn(np)
      common/sacl/sa(nsap)
      common/phi01/p01(np)/phi02/p02(np)/phi03/p03(np)
      common/phi11/p11(np)/phi12/p12(np)/phi13/p13(np)
      common/phi21/p21(np)/phi22/p22(np)/phi23/p23(np)
      common/phii1/pi1(np)/phii2/pi2(np)
      common/phii3/pi3(np)/phii4/pi4(np)
      common/phi4/e(np2p)/phi5/f(np2p)
      common/bndvls/bc0p,bc0p1,bc1p1,bc1p,bc1,bc0
      common/errcl/err(np2p)
      common/serrcl/amxer1,amxer2,ermax
      common/dum1/d1(np2p)/dum2/d2(np2p)
c
c     commons for spline integrals
c
c#### ssi(np2,i),dsi(np2,7,j),tsi(np2,49,k)
c#### i=number of single integrals
c#### j=number of double integrals
c#### k=number of triple integrals
c
      common/ssicl/ssi(np2p,nsip)
      common/dsicl/dsi(np2p,7,ndip)
      common/tsicl/tsi(np2p,49,ntip)
c
      common/nssip/nsi,nsj(nsip),nsv(2,nsip)
      common/ndsip/ndi,ndj(ndip),ndv(4,ndip)
      common/ntsip/nti,ntj(ntip),ntv(6,ntip)
c
      equivalence (sa,sa3)
      dimension sa3(4,4,np2p)
c
c
      fi3(a3,a2,a1,a0,z)=z*(a3+z*(0.5*a2+z*(a1/3.0+z*0.25*a0)))
      fix(a3,a2,a1,a0,z)=z*z*(0.5*a3+z*(a2/3.0+z*(0.25*a1+z*0.2*a0)))
      kk=1
      do 10 j=1,m
      v(j)=0.0
      do 11 k=kk,n
      if(x(j).le.r(k)) go to 12
   11 continue
   12 l=max0(2,k)-i+3
      kk=k
      if(l.lt.1.or.l.gt.4) go to 10
      idx=4*l+16*i-16
      v(j)=(sa(idx-3)+x(j)*(sa(idx-2)+x(j)*(sa(idx-1)+x(j)*sa(idx))))
   10 continue
      return
      entry spvlp(i,x,v,m)
      kk=1
      do 20 j=1,m
```

```
      y(j)=0.0
      do 21 k=kk,n
      if(x(j).le.r(k)) go to 22
   21 continue
   22 l=max0(2,k)-i+3
      kk=k
      if(l.lt.1.or.l.gt.4) go to 20
      idx=4*l+16*i-16
      y(j)=sa(idx-2)+x(j)*(2.0*sa(idx-1)+3.0*x(j)*sa(idx))
   20 continue
      return
      entry spvlpp(i,x,y,m)
      kk=1
      do 30 j=1,m
      y(j)=0.0
      do 31 k=kk,n
      if(x(j).le.r(k)) go to 32
   31 continue
   32 l=max0(2,k)-i+3
      kk=k
      if(l.lt.1.or.l.gt.4) go to 30
      idx=4*l+16*i-16
      y(j)=2.0*sa(idx-1)+6.0*sa(idx)*x(j)
   30 continue
      return
      end
      SUBROUTINE JOBTIME(TENMILI)
      INTEGER LISTITEM(3),TENMILI
      INTEGER SYS$GETJPI,STATUS
C
      LISTITEM(1)= 1031*2**16+4
      LISTITEM(2)= %LOC(TENMILI)
      LISTITEM(3)= 0    !    %LOC(LCPUELAPSED)
      STATUS= SYS$GETJPI(,,,LISTITEM,,,)
C
      RETURN
      END
      FUNCTION GETIME(DUM)
      INTEGER TENMILI
      CALL JOBTIME(TENMILI)
      T=FLOAT(TENMILI)/100.
      GETIME=T
      RETURN
      END
$
```


END
FILMED
6-85